CONTROL OF LIQUID SIMULATIONS

A Thesis
Presented to
The Academic Faculty

by

Karthik Raveendran

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
December 2014

CONTROL OF LIQUID SIMULATIONS

Approved by:

Professor Karen Liu, Committee Chair
School of Interactive Computing
Georgia Tech

Professor Greg Turk, Advisor
School of Interactive Computing
Georgia Tech

Professor Jarek Rossignac
School of Interactive Computing
Georgia Tech

Assistant Professor Nils Thuerey
Department of Informatics
Technical University of Munich

Assistant Professor Chris Wojtan
Department of Computer Science
IST Austria

Date Approved: 29 July 2014

To my parents,

Mallika and Ravee.

ACKNOWLEDGEMENTS

This dissertation would not have been possible without a number of wonderful people.

First of all, I want to thank my parents, Mallika and Ravee. You have always being there for me, supportive of all the decisions that have led me along this path to graduate school. Thank you for your unwavering faith and confidence in me.

Next, to my advisor Greg Turk: I've have been very fortunate to have had you as my mentor over the last five years. You have given me the confidence and freedom to tackle problems that I might otherwise have never considered. Thank you for your patience, kindness and enthusiasm. I will always cherish the time I spent working with you in graduate school.

I'm extremely grateful to my collaborators Nils Thuerey and Chris Wojtan. Greg jests that I have not one, but three advisors and I think that's much closer to the truth. Nils: thanks for figuring out a way to get me to visit Munich — that summer made a huge difference! Chris: thank you for your personal advice over the years and for showing me how much fun it could be to work on fluid simulations.

One of the best things about the graduate program at Georgia Tech is that you get a chance to interact with some incredibly talented and gracious people. To the professors who have been part of my thesis and qualifying committees — Karen Liu, Jarek Rossignac and Blair MacIntyre: you have each had a profound influence on my development and education as a researcher. Thank you for your advice and support.

To my labmates in the Graphics group at Georgia Tech — Mark Luffel, Jie Tan, Sumit Jain, Topraj Gurung, Yuting Ye, Yunfei Bai, Kristin Siu and Alex Clegg: SIGGRAPH deadlines would not have been the same without all of you around!

I'm also grateful to the researchers I have met during my internships at Rhythm

and Hues and at various conferences over the years: Lily Kharevych, Sanjit Patel, Jerry Tessendorf, Tae Yong Kim, James O'Brien, Miguel Otaduy, Adam Bargteil and Adrien Treuille. I really appreciate the time each of you took to give me advice and to discuss various ideas with me.

To my aunt and uncle, Radha and Vasu Ratnam: you made me feel at home in Atlanta. Thanks for all the food and home-brewed beer!

Finally, thanks to my friends who made Atlanta so much fun — Arya Irani, Yee Chieh Chew, Liam MacDermed, Matt Bonner, Heather Munoz, Jonathan Scholz, Jeremy Brudwick, David Wu, Arjun Khurana, Catherine Midkiff, Daniel Gaviria, Diana Ascarrunz, Yacin Nadji, Prateek Bhakta, Brendan Dolan-Gavitt, Ying Xiao, Catherine Grevet, Mariam Asad, Maia Jacobs, Jason Davis and Amy Giedraitis. I'm sorry if I have missed out anyone here,

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Over the last decade, advances in fluid simulation and rendering have helped animators synthesize photorealistic shots for movies that would have been virtually impossible to create by manually animating the liquid. Despite the advent of these computational methods, fluid simulation in movie production still involves a large degree of trial and error. One of the main reasons for this is the highly non-linear and dynamic nature of water, which can cause it to quickly deviate from the initial conditions prescribed by the animator.

In this dissertation, we propose a set of techniques for creating animations of liquid that meet desired artistic criteria without the customary tuning of numerous physical parameters. The basis for our work is the mesh-based representation of the liquid surface which lends itself to efficient algorithms that can control the output of simulations.

First, we show how an animator can create animated characters and shapes that behave as if they were made of liquid using our mesh-based control method. Our approach allows for multiple levels of control over the simulation, ranging from the overall tracking of the desired shapes to highly detailed secondary effects.

Next, we present a novel technique for interpolating between fluid simulations with free surfaces. We construct 4D spacetime meshes from animations and register them using a non-rigid ICP algorithm. By incorporating user input to align visually important regions, we can produce plausible animations that look like a blend of the two input sequences and can handle animations with changes in topology, all without re-simulating the fluid. We demonstrate how this could have applications in pre-visualization and video games.

CHAPTER I

INTRODUCTION

1.1 Motivation

Over the last decade, advances in fluid simulation and rendering have helped animators synthesize photorealistic shots for movies that would have been virtually impossible to animate using manually specified keyframes. Current techniques can efficiently simulate a wide gamut of fluid behavior, ranging from small scale effects such as swirling and sloshing of water in a glass, to massive natural phenomena such as turbulent rivers and crashing ocean waves. Despite the advent of these computational methods, there is a large degree of trial and error involved in the process of creating water effects for films. In certain cases, animators have opted to eschew the use of a simulation altogether in favour of tools like particle systems.

One of the main reasons for this is the highly non-linear and dynamic nature of liquids, which can cause them to quickly deviate from the initial conditions prescribed by the animator. In practice, this means that the animator has to sample the simulation parameter space (by trying various combinations of initial velocities, viscosity, boundary conditions such as positions of objects) to get a sense of the range of outputs that the simulations can produce.

Unfortunately, liquid simulations are generally computationally expensive and require hours of processor time to produce results. This makes the process of narrowing down the exact parameters fairly tedious and time-consuming. To work around this, artists usually run low resolution versions of simulations in the early stages and switch to the high resolution counterparts closer to final production. A related issue that complicates this workflow is that liquid simulations at different resolutions do not

generally produce qualitatively similar outputs for the same set of initial parameters.

In many cases, the liquid simulation is related to the motion of other objects or characters in the scene. For instance, in the production of the film Surf's Up, the animators needed to create consistently key-framed animations of characters that rode on waves and opted for a procedural wave system over a fluid simulation even though the latter would have produced a more realistic surface [28]. Similarly, during the production of Ratatouille, animators chose to model splashes using NURBS surfaces due to the difficulty of directing a detailed fluid simulation. As the authors point out in [27] – "A simulated splash that went one quarter of an inch higher than the previous iteration could mean the difference between a composition that works and one that fails". How then can we bridge this divide between the controllability of hand-tuned animation and the realism and physical fidelity of a simulation? It is instructive to look at some of the tools employed in production to tame the output of simulations. Some of these include virtual force or velocity fields that can guide the flow of liquid in a region, sources to pump water into a scene, sinks to drain excess water and invisible objects to ensure that water is restricted to a certain region. In Rasmussen and colleagues describe techniques for manipulating level-set based simulations using parameters like viscosity and isosurface values [104].

This overarching concern about the predictability of water simulations is one of the motivating factors behind the research presented in this work. **The goal of this dissertation is to explore algorithmic approaches to controlling liquid simulations such that the resulting outputs match the expectations of the animator. Liquids can be effectively controlled both during and after simulation by using a mesh-based surface representation**. We contend that by focusing on the boundary of the liquid which separates it from air or solids, rather than its volume, we can produce animations that are easy to specify and predict.

1.2 Desirable qualities in a control algorithm

Before we begin to develop algorithms for liquid control, it is useful to understand some of the attributes that are desirable in them. In this section, we look at four qualities that one might find in an algorithm - the scenarios to which it is best suited, the ease of specifying outputs, its ability to match these criteria, and its computational cost.

One possible characterization of control algorithms is in terms of the *naturalism* of the phenomenon to which they are being applied. For instance, consider animating shapes or creatures made out of water as seen in movies like the Chronicles of Narnia : Prince Caspian or Avatar: The Last Airbender. It becomes immediately apparent that water needs to be artificially moulded into these shapes and that it is highly unlikely that a simple set of initial conditions would result in the desired output. Further, it is not adequate to merely force water to form the shapes; we also desire the dynamic effects that we typically associate with water such as splashes, dripping streams, surface waves and ripples. On the other hand, if we focus on realistic phenomena such as splashes or waves, we might be more interested in being able to precisely govern the timing or the size of a certain surface feature while leaving the rest to the simulation.

Another axis along which these algorithms can be evaluated is their ability to match certain conditions or *constraints* that have been prescribed by the user. These could be in the form of the surface acquiring a particular shape, or the velocity of a certain region of liquid matching a specified vector field. Ideally, a control algorithm will apply no more force than is required to produce a certain effect. Still, it is often useful to have the possibility of relaxing these constraints so that the motion looks more natural. Further, it is vital that the algorithm fails gracefully when it is unable to achieve the desired result.

From the perspective of usability, a control algorithm must not impose a *computational overhead* on the simulation that outweighs its utility. Considering that simulations typically take on the order of hours to run, and that in most cases the computational bottleneck lies in the pressure solve stage of a liquid simulation, this is less of a restriction than it might seem at first glance.

Perhaps more importantly, the *practicality* of an algorithm comes down to the ease of specifying the desired outputs. In certain cases such as creating animated characters made of water, it might be simpler for the animator to provide a set of target shapes or keyframes that must be matched by the liquid surface. However, this might be too restrictive for complex scenes and one might want to specify only a portion of the surface. At the other end of the spectrum, one might prefer to take a less prescriptive approach and instead make subtle modifications to the output of an existing simulation.

1.3 Control algorithms

In this dissertation, we discuss two approaches to fluid control:

1.3.1 Controlling liquids using meshes

In Chapter 4, we present a technique that favours artistic goals over more naturalistic phenomena. We introduce a new method for creating high quality fluid animations that provides the animator with multiple levels of control over the simulation while meeting the basic design goals for the bulk motion of the liquid. Our method takes in a dense sequence of *control meshes* as its input. This animated mesh sequence can either be directly provided by the animator or can be generated from a sparse set of user-defined input meshes using our volume-preserving morphing technique. The animator can then run the simulator and adjust some simple dials that select the *drippiness* or *looseness* of the water surrounding the target shape. In addition, the animator can add specific surface details such as bumps or introduce surface waves to

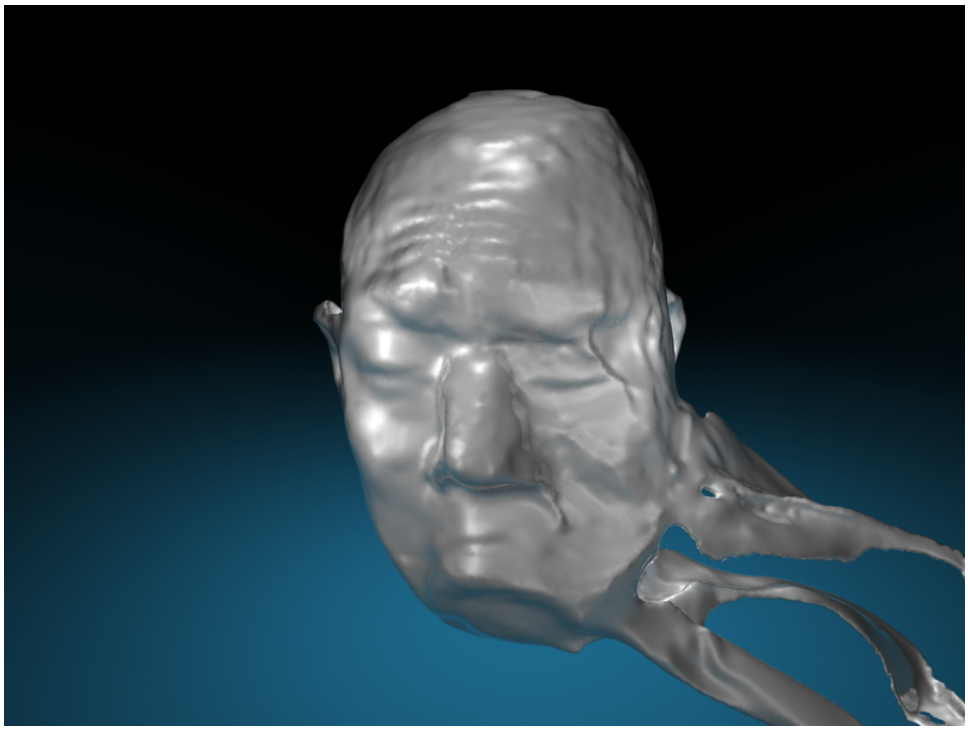

Figure 1: A rendered frame from an animation produced using our mesh-based surface control algorithm.

create a more dynamic look while still matching the motion prescribed by the given control meshes.

Our scheme clearly divides control over the basic underlying motion from that of secondary effects of the fluid and this makes it intuitive for the animator to fine-tune specific aspects of the animation. Unlike previous approaches to this problem, we can control details of the liquid surface at scales smaller than a single grid cell. This freedom allows us to obtain high fidelity effects at reasonable grid resolutions.

1.3.2 Blending liquids

In Chapter 6, we present a method for smoothly blending between existing liquid animations. This form of liquid control is suited for realistic scenarios where the animator might want to make small changes to the initial conditions. We introduce a semi-automatic method for matching two existing liquid animations, which we use to create new fluid motion that plausibly interpolates the input. Our technique can

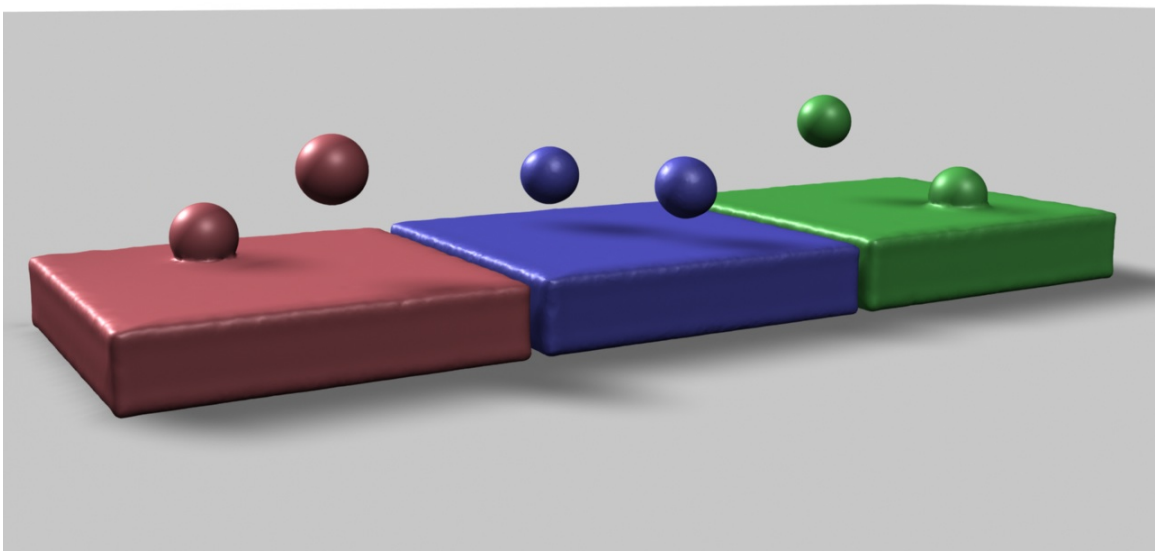

Figure 2: A rendered frame from an animation produced using our blending algorithm. The middle image is the 50 percent interpolant between the left and right animations.

be used to instantly create hundreds of new simulations, or to interactively explore complex parameter spaces. Our method is guaranteed to produce output that does not deviate from the input animations, and it generalizes to multiple dimensions. Because our method runs at interactive rates after the initial precomputation step, it has potential applications in games and pre-visualization.

1.4 Contributions

The algorithms presented in this dissertation make several contributions to the field of computer graphics and computer animation. Specifically, these contributions are:

- **A volume-preserving morphing algorithm**: We introduce an optimization framework that produces a dense sequence of meshes from a sparse set of user defined meshes. Our algorithm supports positional and velocity constraints at these user-defined keyframes. For moderate deformations, the generated meshes conserve the original volume while changing shape until they meet the

final target.

- **A multi-level control technique**: We show that we can efficiently direct liquids by splitting up control forces between the Eulerian simulation grid and a high-resolution surface mesh. We compute control forces on the grid using the velocity of the control meshes as boundary conditions on the pressure solve. This allows us to not only reproduce the bulk flow from the keyframes, but also allow for natural inertial effects such as dripping or separating sheets of water from a fast moving shape. Further, we can generate highly detailed water surfaces and effects by using a mesh-based representation of the surface that is attracted to the user-provided control meshes.

- **Interpolation between free surface fluid simulations with changing topologies**: We present an algorithm for interpolating between two water animations by using a spacetime representation and show how this idea can be extended to deal with multiple inputs.

- **A 4D non-rigid ICP algorithm**: We extend the non-rigid ICP algorithm to operate with 4D spacetime meshes and also show how we can incorporate user guidance through the form of various soft and hard constraints.

- **A subsampling technique for efficient registration of meshes with millions of vertices**: We describe a subsampling algorithm that makes it feasible to register extremely large meshes without severe memory requirements.

- **A fast surface extraction algorithm**: We show how we can extract 3D triangle meshes from a 4D space-time surface at interactive rates.

CHAPTER II

BACKGROUND

This dissertation is inspired by numerous works of research in computer graphics and animation. In this chapter, we begin by tracing the key developments in the area of fluid simulation and then move on to a comprehensive discussion of algorithms and techniques used to control fluids. We then briefly outline techniques used to register meshes and conclude with a discussion on spatio-temporal representations of animations.

2.1 Fluid simulation

Fluid simulation has become an established field of computer graphics research, after the seminal work of Foster and Metaxas [39] and the popularization of stable advection routines [119]. In addition to such Eulerian approaches, the state-of-the-art is steadily advanced by purely particle-based [87; 117] and hybrid approaches [142].

The work presented in Chapter 4 draws upon commonly used methods for animating fluids in graphics. Specifically, we perform fluid simulation using an Eulerian framework on a staggered MAC grid [39]. We enforce fluid incompressibility by solving a Poisson equation [120]. Details of these techniques can be found in the work of Bridson [23]. For fine surface details, we make use of a mesh-based surface tracker [137], and we carry out wave dynamics on surfaces using the methods of Thurey et al. [130].

Our work using spacetime surfaces in Chapter 6 is oblivious to the particular algorithm employed for solving the Navier-Stokes equations. However, the choice of a surface tracker has implications on what kind surface data is generated by the simulation. A popular class of surface tracking algorithms is based on level-sets [93], and particle

level-sets [33]. Another class of methods uses an explicit surface representation [25; 137; 84]. These methods better preserve fine details, but they require additional work to retain well-shaped elements and handle changes in topology. For particle based methods a variety of surface reconstruction approaches have been proposed, e.g., using anisotropic kernels [140]. While we use a surface tracker as described by Wojtan et al. [137], our method could be similarly applied to any other surface tracker, as long as a triangle mesh is generated for each frame of the animation.

2.2 Fluid control

Foster and Metaxas [41] introduced the notion of fluid control to computer graphics through the use of embedded controllers. These controllers governed attributes such as fluid properties, pressure fields, velocity fields and boundary conditions while maintaining physical correctness. Since then, researchers have looked at expanding fluid control in various directions such as manipulating velocity fields, closely tracking user-specified keyframes and ensuring that high resolution simulations match their low resolution counterparts.

2.2.1 Key-framed control

Fattal and colleagues proposed a method that controls smoke to form and target shapes using a combination of a forcing term as well as a gathering term to counter the diffusive nature of smoke [36]. Their control algorithm does not require an optimization framework to form the provided shape targets, however the exact timing of the animation cannot be controlled.

Shi and Yu approached this problem using implicit functions to denote the current and target shape and used the signed distance field to perform a shape matching between the two. By imposing approriate velocity constraints, they were able to get smoke to follow moving targets set by the animator [111].

Instead of applying implicit functions for control, Hong and Kim solved for a

geometric potential field that takes the current shape to the desired target [53].

Shi and Yu introduced a new technique for liquids with free surfaces where a proportional derivative (PD) controller was used to track rapidly changing targets [112].

Our control algorithm described in Chapter 4 shares certain similarities with these control methods in that it also uses Neumann boundary conditions on an Eulerian grid as a means of applying control forces. However, due to the construction of our boundary conditions, we can achieve close tracking without the need for a carefully tuned PD-controller.

McNamara et al. proposed a novel method of dealing with keyframed control of fluids [83]. They treat a fluid simulation as a composition of functions and minimize an objective function to solve for the control forces. For reasons of efficiency, they applied the adjoint method to compute derivatives from the simulation with respect to each control parameter. However, this technique still requires the calculation and storage of these derivatives for the entire simulation at each timestep and as a consequence, it is computationally expensive, particularly at high grid resolutions that are necessary for capturing details.

2.2.2 Modifying velocity fields

A popular way of affecting the motion of a fluid is to directly modify the velocity field. Since naive modifications can result in unphysical behavior, researchers have sought to constrain the way the animators can manipulate these fields while trying to retain most of the power and flexibility of this type of control.

Cheney simplifies the problem of editing a velocity field by breaking it up into a set of tiles, each of which represents a particular flow field. These tiles can then be stitched together to create a more complex flow. However, Pighin and colleagues argue that the Eulerian representation of velocities on a regular grid is not convenient

for editing flows and instead propose a spatio-temporal representation by using radial basis functions centred along pathlines. Users can then manipulate these points on the pathlines as if they were handles on a deformable object. However, this work focuses on smoke and does not deal with free surface liquids.

Rasmussen et al. [104] and Thuerey et al. [129] approach this problem by using control particles to adjust velocities, viscosity and other properties of the liquid. These techniques avoid aliasing artifacts on the grid by using this form of Lagrangian control. However higher level goals such as resembling particular shapes need to be translated to quantites that can be carried by these particles. Instead of using a particle levelset [104], we use a Lagrangian surface tracking technique with a coarse Eulerian simulation to handle fine details in our control algorithms.

Another option is to entirely bypass the simulation step by using a procedural velocity field. Procedural generation has been a mainstay of computer graphics and its introduction to fluids has led to widespread adoption in both academia and industry. While numerous researchers applied Fourier synthesis [113; 121] and Perlin noise [101] to generate velocity fields, these techniques were not sufficiently specific to create divergence-free fields and did not respect boundary conditions. Building on these ideas, the work of Kniss and Hart, Patel and Taylor, and Bridson solved these two problems and provided artists with a broader palette of methods to produce highly detailed and realistic flows [32; 99; 24].

2.2.3 Detail enhancement

While procedural generation is fast and intuitive, animators still resort to fluid simulators to produce more dynamic and realistic animations that react to the environment. However, as we have discussed earlier, these simulators are notorious for producing noticeably different outputs for even minor perturbations in the initial conditions. An animator may not want to modify the overall flow of the fluid but might instead wish

to enhance the visible details such as swirling of smoke and waves on the surface of a liquid. Generally, these methods are based on a Reynold decomposition of the velocity field of a fluid resulting in a mean velocity field and a more rapidly changing field [89; 66]. The challenge then is to synthesize this changing flow field such that it respects the Kolmogorov spectrum and integrate it with the larger scale flow. We will not delve into these methods in detail in this dissertation. However, they indicate possible ways of adding detail to the output of our simulations.

2.2.4 Guiding simulations

A recent class of methods in fluid control [91; 56] has aimed to make higher resolution simulations loosely match the behavior of their lower resolution counterparts with the same initial conditions. These are intended for creating production quality output while allowing the artists to prototype using faster, low resolution simulations and are targeted at naturalistic scenarios.

A slightly different approach involves using a high resolution simulation but forcing it to obey certain boundary conditions or user specified constraints and thereby restricting possibly divergent behaviours [90; 94]. Our work in Chapter 4 shares certain commonalities with that of Nielsen and Bridson [90] in terms of the potential flow solution that used to compute interior velocities, however our focus is to generate more supernatural phenomena such as creatures made of water and in particular, on highly detailed yet controllable liquid surfaces.

2.2.5 Reduced models of fluids

The final category of algorithms we consider in this section are based on the principle of model reduction. Complex systems with non-linear dynamics can be approximated by lower dimensional models by projecting them onto a predetermined basis. This basis is typically constructed by variants of Proper Orthogonal Decomposition (POD) which involves picking a set of snapshots of the fluid and then computing the desired

number of eigenvectors (sorted by their eigenvalues in descending order) as seen in the works of [106; 131]. The advantage of model reduction is that it can be used for fast and efficient control because these algorithms operate in a significantly lower dimension compared to that of the original simulation.

Unfortunately, liquids with free surfaces have not been amenable to this class of methods for a couple of reasons. Firstly, the free surface imposes additional boundary conditions and the fast marching method that is used to compute the new signed distance field is not a linear operator that is suited to existing reduction techniques. Secondly, a very rich basis is necessary to capture the behaviour of liquids due to numerous changes in topology. This remains an avenue for future research.

2.3 Registration

In this section, we cover methods that relate to the work on blending liquids presented in Chapter 6. There is a rich literature of work relating to the registration of surfaces and point clouds in both computer vision and computer graphics. The iterated closest point method (ICP) was introduced by Besl and McKay [17] and numerous improvements have been suggested over the years [50; 43; 107; 26]. Our work is closely related to the non-rigid ICP algorithms [4; 79]. This technique has been applied to problems such as completion of dynamic shapes [80] where they reconstruct a temporally coherent and water-tight sequence of meshes from data captured by multi-view acquisition systems. Non-rigid ICP has also been applied to the tracking of shapes with changing topology such as liquid surfaces [20]. Unlike these methods which register a pair of 3D meshes, we extend the non-rigid ICP algorithm to register entire animations of liquids.

During each iteration of the ICP algorithm, we deform the source animation to move all vertices towards their current correspondences. We perform this deformation using a variation of the embedded deformation model first introduced by Sumner and

colleagues [124] and extended by Li and colleagues [79]. Our deformations are applied in 4D spacetime and we do not impose any non-linear constraints on the columns of the affine transformation matrix at each node. This leads to a simple weighted linear squares solve that can be solved using standard methods.

Aside from registration with ICP, researchers have explored a variety of other approaches for matching surfaces. Szeliski et al. proposed to use splines [125], while others used local similarity transforms [95], intrinsic features [44], probabilistic correspondence generation [8] and high-order graph matching [141].

Numerous methods in computer graphics morph between surfaces of different topology using an implicit representation [31; 22; 132]. These methods are convenient because they do not require explicit correspondence generation or advanced meshing techniques. On the other hand, they are unable to use correspondences to align salient features. For this reason, we opt to use ICP to handle the majority of the registration, but our approach could be combined with methods such as those using implicit representations for registering additional small-scale features.

2.4 Animations as structures in spacetime

The idea of treating animation data as a surface or volume in a higher dimensional space has been used by graphics researchers for various purposes. A theoretical foundation for viewing animations as Cartesian products in spacetime was introduced in Skapin et al. [116]. Klein and colleagues applied this notion to frames of a video sequence and created a rendering solid that could be used to render stylized videos [67]. Pasko and colleagues used spacetime volumes for morphing between objects [98]. Auburt and Bechmann demonstrated how spacetime objects could be deformed to produce animations involving topological modifications [9]. Kwatra and Rossignac applied compression algorithms to the bounding triangles of a space-time volume of 2D cel animation data [70]. Schmid and colleagues construct a time aggregate object

by connecting triangle meshes in time and use this data structure to produce stylized blurring and stroboscopic images [109]. Similarly, our algorithm creates a spacetime mesh per animation by linking up frames of a liquid surface. However, we also need to consider changes in connectivity and topology between frames due to the evolving fluid surface and we handle these with tetrahedra instead of bilinear patches.

CHAPTER III

VOLUME-PRESERVING MORPHING

3.1 Definitions

In-betweening (or tweening for short) is defined as the process of producing intermediate frames between two specified frames of animation (keyframes) such that the transition looks seamlesss and plausible. A closely related idea is that of morphing which smoothly changes an image or a shape into another. While morphing could be viewed as an application of tweening (where the in-between frames are rendered), for the purposes of our discussion, we will consider the two terms to be interchangeable.

3.2 Motivation

One of the key benefits of morphing is that it reduces the amount of effort required from an animator. One can produce a animation simply by specifying a sparse set of keyframes. The challenge then lies in ensuring that the resulting motion is smooth and that the intermediate shapes are visually acceptable. Traditionally, morphing relies on having good correspondences between the source and target shapes such that salient features are preserved during the transition. For instance, when morphing between two human faces, it is often desirable to map important characteristics such as the eyes, nose and lips on both shapes so that the intermediate shapes also resemble faces.

Even when we have perfect correspondences between the two shapes, it might be desirable to impose additional restrictions on the in-between shapes. For instance, constraints on the position or the velocity of a subset of vertices could be used to change the timing of the transition. One could also enforce a global constraint where the intermediate shapes are forced to conserve volume. Simple techniques such as

linear interpolation between corresponding vertices can result in shapes with significantly altered volumes (see Figure 3).

While volume-preservation is a generally desirable visual property of an animation, it takes on special signficance when it comes to morphing shapes that are made of water. Firstly, water is a nearly incompressible fluid which means that the shapes cannot change their volume unless there is a source or sink. Further, if we use these intermediate shapes in a traditional fluid simulator, we would be effectively adding a net divergence to the system which can result in artifacts such as sudden bursts of water or dramatically altered velocities. In this chapter, we describe a technique for performing volume-conserving morphs between shapes.

3.3 Volume-preserving morphing

The input to our morphing algorithm is a set of control meshes $\boldsymbol{K}$ defined by closed, manifold, connected triangulated meshes at user specified instants of time t: $\boldsymbol{K} = \{K_t\}$. We require that the vertex correspondences and the connectivity of the mesh are preserved between these meshes. The user also needs to specify the number of in-between frames T to be generated between each pair of given meshes.

The output of the algorithm is a set of in-between triangulated meshes M_t. Since the connectivity of the mesh is assumed to be fixed, each in-between frame M_t is fully defined by the positions of the V vertices of the mesh, $\boldsymbol{x}_i^t$ where each $\boldsymbol{x}$ denotes a position vector, t is the instant of time and i is the vertex number.

The goal of the morphing algorithm is to produce a motion that is as smooth as possible while conserving volume. Specifically, if we assume that each K_t has the same solid volume, then we want each of the in-between meshes M_t to preserve that solid volume. This amounts to minimizing the following smoothness energy function (i.e. avoid sharp changes in velocity):

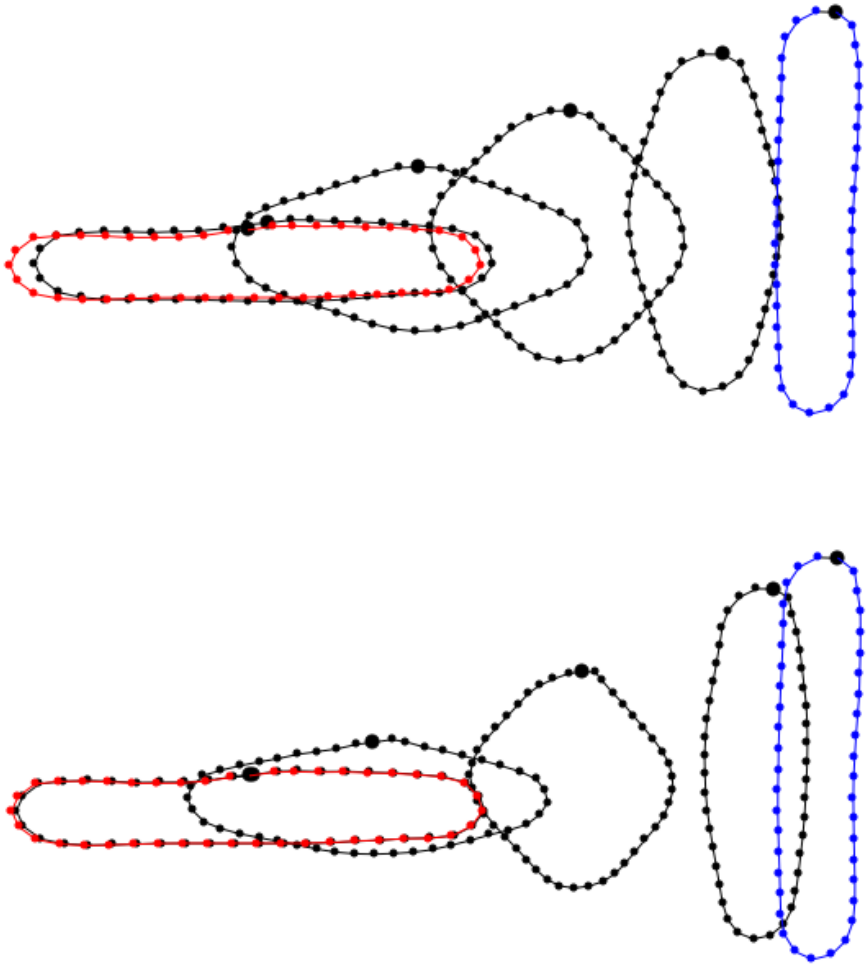

Figure 3: Comparison of linear interpolation (top) with our volume preserving morph (bottom). There is a volume gain of more than 50% in the middle shape for the linear morph.

$$F_{smooth} = \sum_{t=1}^{T-1} \sum_{i=1}^{V} \|\boldsymbol{v}_i^{t+1} - \boldsymbol{v}_i^t\|^2 \text{ where } \boldsymbol{v}_i^t = \frac{\boldsymbol{x}_i^{t+1} - \boldsymbol{x}_i^t}{\Delta t} \tag{1}$$

subject to the volume constraint:

$$\text{Volume}\left(M_t\right) = \text{Volume}\left(K_0\right), \forall t \in T \tag{2}$$

Note that this constraint is defined over the entire volume while our input meshes are surfaces bounding a volume. If we take the time derivative of Equation 2, we have

the following equation:

$$\frac{\partial \text{Volume}\,(M_t)}{\partial t} = 0,\ \forall t \in T \tag{3}$$

Since we are assuming that the meshes are made of incompressible material, this implies that the flux through the mesh must be zero. We know from the divergence theorem that:

$$\iiint_V (\nabla \cdot \mathbf{F})\ dV = \oint_S (\mathbf{F} \cdot \mathbf{n})\, dS. \tag{4}$$

$$\text{where } \mathbf{F} = \rho \mathbf{v} \tag{5}$$

Applying this to Equation 3, we obtain a transformed constraint that is purely defined using quantities on the surface of the mesh:

$$\sum_{i=1}^{F_t} \left(\boldsymbol{v}_{\boldsymbol{f}_i}^{\,t} \cdot \boldsymbol{n}_i^t \right) S_i = 0,\ \forall t \in T \tag{6}$$

$$\text{where } \boldsymbol{v}_{\boldsymbol{f}_i}^{\,t} = \frac{\sum_{j=1}^{3} \boldsymbol{v}_j^t}{3}$$

Here, $\boldsymbol{v}_{\boldsymbol{f}_i}^{\,t}$, $\boldsymbol{n}_i^t$ and S_i denote the velocity, normal and area of the triangular face f_i and F_t denotes the number of triangular faces at time t respectively. We compute velocities using forward differences between vertex positions. Unfortunately, the divergence constraint in Equation 6 is non-linear because $\boldsymbol{n}_i^t$ and S_i depend on the vertices of the triangle. Further, our optimization is fairly high dimensional with $T \times 3V$ variables and this makes it extremely difficult to obtain a global minimizer unless we begin with a very good initial guess.

In order to make this computation tractable, we linearize Equation 6 by approximating the normal and area at time t by interpolating between the known quantities at the the start and end control meshes defined by the user. We use a simple linear interpolation for the area of the triangle, but for the normal, we perform a spherical

interpolation between the quaternionic representations of the start and end normals. This makes the equation linear in velocities, and consequently, in vertex positions. In practice, this approximation for the normals turns out to be a good starting point and can be adjusted over multiple iterations.

We can introduce a Lagrange multiplier λ_t per constraint, and obtain a new objective function that needs to be minimized with respect to $\boldsymbol{x}_i^t$ and maximized with respect to λ_t:

$$F_{new} = \sum_{t=1}^{T} \sum_{i=1}^{V} \|\boldsymbol{v}_i^{t+1} - \boldsymbol{v}_i^t\|^2 + \sum_{t=1}^{T} \lambda_t \left(\sum_{i=1}^{F} \boldsymbol{v_f}_i^t \cdot \boldsymbol{n}_i^t S_i \right) \tag{7}$$

By taking the partial derivatives with respect to each free variable and setting them to zero, we obtain a system of linear equations in $\boldsymbol{x}_i^t$ and λ_t, with a total of $T \times (3V + 1)$ variables. This results in a sparse symmetric indefinite linear system (Equation 8) that is well studied in linear algebra.

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} \tag{8}$$

There are many ways to solve this linear system such as LU or LDL factorizations of the matrix ([46]), null space methods that eliminate the constraints ([47]) or approaches that rely on the range space ([45]). We use a range-space method by computing the Schur complement of the matrix. The Schur complement transforms the problem of solving a $(m+n) \times (m+n)$ system into one that requires inverting two smaller matrices ($m \times m$ and $n \times n$). This is particularly useful when one of the smaller matrices has some property (such as sparsity or symmetric positive definiteness) that permits the fast computation of a inverse.

The solution to our linear system can be computed as follows:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} \tag{9}$$

$$\lambda = \left(\mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\left(\mathbf{B}^T\mathbf{A}^{-1}\mathbf{p} - \mathbf{q}\right) \tag{10}$$

$$\mathbf{x} = \mathbf{A}^{-1}\left(\mathbf{p} - \mathbf{B}\lambda\right) \tag{11}$$

We need to compute the inverse of the matrix $\mathbf{A}$ in order to proceed with the Schur complement solve. Note that each $\boldsymbol{x}_i^t$ appears in three terms of the smoothness energy ($\|\boldsymbol{v}_i^{t+1} - \boldsymbol{v}_i^t\|^2$, $\|\boldsymbol{v}_i^t - \boldsymbol{v}_i^{t-1}\|^2$ and $\|\boldsymbol{v}_i^{t-1} - \boldsymbol{v}_i^{t-2}\|^2$). If the elements of the vector $\mathbf{x}$ are ordered sequentially in time for each vertex, we obtain a symmetric five-band stencil due to the forward difference discretization:

$$\mathbf{A} = \begin{bmatrix} 12 & -8 & 2 & & & \\ -8 & 12 & -8 & 2 & & \\ 2 & -8 & 12 & -8 & 2 & \\ & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

This causes $\mathbf{A}$ to be sparse, symmetric and positive definite. $\mathbf{A}$ also has the additional property that it is block-diagonal because adjacent vertices at an instant of time are not related to each other in the smoothness energy function:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A_0} & \mathbf{0} & \cdots & \\ \mathbf{0} & \mathbf{A_1} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{0} & \ddots & \vdots \end{bmatrix} \tag{12}$$

As a consequence, it can be easily shown that the inverse of $\mathbf{A}$ is

$$\mathbf{A^{-1}} = \begin{bmatrix} \mathbf{A_0}^{-1} & \mathbf{0} & \cdots & \\ \mathbf{0} & \mathbf{A_1}^{-1} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{0} & \ddots & \vdots \end{bmatrix} \tag{13}$$

We compute the Cholesky decomposition of each unique $\mathbf{A_i}$ using the Cholesky-Banachiwicz algorithm ($\mathbf{A_i} = \mathbf{L_i}\mathbf{L_i}^T$). This yields a lower triangular matrix $\mathbf{L_i}$ that has the same sparsity and structure as the matrix $\mathbf{A_i}$ and can be computed in $O(n)$ instead of the typical $O(n^2)$ since there are exactly three non-zero elements per row in the decomposition. Finally, the inverse of $\mathbf{A_i}$ can be computed by first solving for the inverse of $\mathbf{L_i}$ to obtain $\mathbf{L_i}^{-1}$ and then performing a simple multiplication with its transpose $\left(\mathbf{L_i}^{-1}\right)^T\mathbf{L_i}^{-1}$. Further, we note that only frames affected by a velocity or position constraint result in a different structure in these submatrices. So in general, we can compute the inverse of a single submatrix and copy it over to other identical blocks in the matrix.

It is interesting to note that $\mathbf{A}$ is completely determined by the number of vertices of the input shapes and the number of in-between frames. As a result, the inverse of $\mathbf{A}$ can be precomputed and reused for a variety of morph targets as long as the number of vertices is kept fixed. Once we have the inverse, we can compute the Lagrange multipliers that enforce the constraints and then solve a $T \times T$ system using LU decomposition to obtain our inbetween positions $\mathbf{x}$. Typically, the number of inbetween frames (T) ranges from 30 to 60 and this does not prove to be a computational bottleneck.

Our basic implementation (without using optimized BLAS libraries) can solve large systems (1,000,000 free variables) in less than 30 seconds without any numerical instabilities. In contrast, iterative solvers such as biconjugate gradient and GMRES did not converge on a solution in a majority of our test cases, while direct LU decomposition became prohibitively expensive for such large matrices.

Note that the normals and triangle areas resulting from this solve may differ from our predicted versions used to construct the linear system. In such cases, we compute a new set of normals and areas from the previous solution and repeat the process until convergence. In practice, this converges using one or two iterations for our test cases

Figure 4: A comparison of the in-betweens generated by our algorithm with (bottom) and without velocity constraints (top). Notice how the higher end velocity constraint causes an initial squashing in order to smoothly hit a higher velocity in the same amount of time.

including those involving large rotations. These iterations are inexpensive because the matrix $\mathbf{A}$ does not depend on the normals or the triangle area and this allows us to reuse its inverse computed at the beginning of the solve. We can also incorporate additional positional or velocity constraints while maintaining volume by introducing more Lagrange multpliers.

We opted to constrain the total divergence at each frame to be equal to zero. In other words, we chose to constrain the derivative of the volume with respect to time, instead of constraining the volume directly. The main reason behind this is that the convergence of the iterative process is much faster with the weaker derivative constraint while a direct volume constraint produces significant oscillations. Our results did not improve when we used the even weaker, second derivative of volume as a constraint and instead, this caused a noticeable loss in volume.

Figure 5: Input meshes to our volume-preserving morpher in the first row, with the in-betweens on the second row. The final simulation sequence that uses the morpher output is shown in Figure 13.

3.4 Results

Figure 3 shows a 2D scenario where our volume conserving morph manages to maintain the original volume while a simple linear interpolation causes an expansion of over 50%. Our morphing technique offers an easy way for animators to produce a smooth deformation between shapes while conserving volume without the use of a skeleton or a character rig. Further, it offers control over the shape by letting the user specify target velocities or positions for parts of the shape through the course of the morph. The biggest benefit is that we can use the resulting velocity field (from finite differences between corresponding vertices) directly inside a fluid simulator and track the animation perfectly since these velocities are not modified by the pressure projection step. This allows the animator to focus on creating the basic motion independent of the simulator.

The bunny twisting sequence in Figure 5 shows the results of using our mesh

morphing for a 3D animation where one keyframe was produced using a Maya deformer on a model of a Stanford bunny and our algorithm generated the in-betweens. Figure 13 shows the results of using these control meshes inside a fluid simulator. Our algorithm also shows interesting results when velocity constraints are specified on certain keyframes. Figure 4 demonstrates a squash and stretch effect when higher velocities are specified at the end frame.

Table 1 shows the numerical results of a comparison against linear interpolation. The source and target were of the same volume in all tests. We see that linear interpolation can lead to large changes in volume while our algorithm stays within 4% of the original.

Note that it is indeed possible to use this formulation to morph between shapes that differ in volume by interpolating the quantity over the inbetween frames and accounting for the change on the right hand side of equation 6.

3.5 Limitations and future work

One of the biggest challenges in a space-time formulation is achieving a reasonable tradeoff between speed and accurate results. In our case, we set up volume-conserving morphing as a space-time problem where the unknowns are the vertices of a mesh at each in-between frame. This forces us to preserve vertex connectivity between

Table 1: Comparison of our volume preserving morphing against linear interpolation. Numbers indicate percentage of original volume.

	Volume Preserving			Linear		
Sequence	Min.	Max.	Avg.	Min.	Max.	Avg.
Bunny Rotate	99.7354	100.329	100.018	90.946	100	94.3362
Bunny Twist	99.9992	100.567	100.159	62.3989	100	75.7682
Bunny Squash	99.9746	100.532	100.197	100	119.24	112.296
Armadillo Stretch	99.7463	103.972	101.094	63.0679	100	76.1993

user-specified keyframes and can be restrictive if the animator wants to morph between meshes that have provided by different sources that do not have vertex correspondences. One possible solution is to perform a non-rigid registration between key-frames in these cases to obtain these correspondences.

Another consequence of the space-time setup is that we had to linearize the volume-preserving constraint in order to avoid getting stuck in local minima. While this turned out to be a good approximation for moderate deformations, one might want an exact solution in certain cases. It remains an open challenge to find efficient algorithms for zeros of large non-linear functions.

Finally, our method only works on water-tight, manifold meshes. If we want to deal with meshes with boundaries, we would have to perform some form of hole-filling before applying our algorithm to them.

CHAPTER IV

CONTROLLING LIQUIDS USING MESHES

4.1 Introduction

In Chapter 1, we discussed several attributes that an animator might want to see in a fluid control algorithm. In particular, the type of liquid animation plays an important role in determining the options available to the animator to tune a simulation. One class of methods attempts to add as little external force as possible to a liquid simulation so that the output meets the animator's expectation while looking natural and unforced. However, when it comes to animating shapes and characters that are made of liquid, it quickly becomes apparent that most liquids are highly unlikely to take on such configurations without extensive manipulation. Our aim then is to first ensure that the resulting animation closely resembles the original intention of the animator, and then expose controls that let the animator dial in how liquid-like it ought to behave.

We begin by assuming that the animator has a set of shape targets that ought to be tracked by the liquid. This is common in production where animated characters have rigs associated with them. Our goal is to take these targets and produce an animation such that it appears that the animated object looks and behaves as if it were filled with liquid. More specifically, the total motion of a controlled liquid can be viewed as the result of the composition of the following forces:

- Inertial forces: These come from the motion of the liquid volume and changes in boundary conditions affect velocities throughout the bulk of the liquid. Visually, inertial effects convey the impression of mass (or volume) and are essential in demonstrating that the character is not merely a shell.

- Viscous forces: Viscosity is a measure of the resistance of a liquid to flowing and comes into play when animating liquids like honey or molten chocolate that are thicker than water.

- Surface forces: Compared to the other forces, these forces act at smaller scales and only affect the liquid surface. Some examples include surface tension, cohesive and adhesive force.

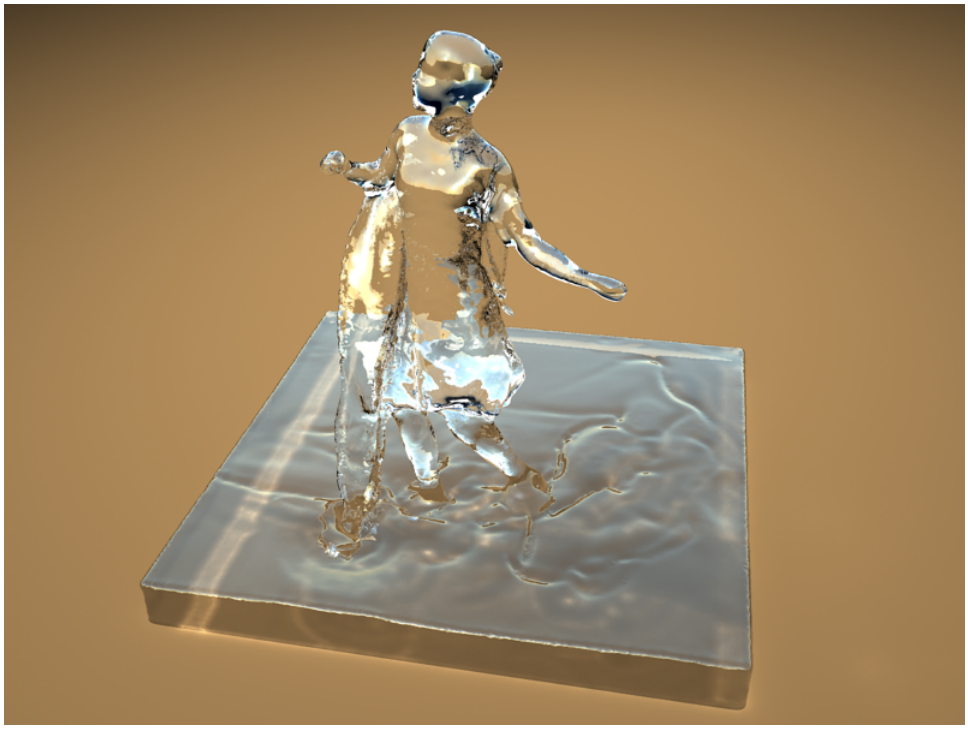

Figure 6: A dancer made of water animated using our control algorithm

In this chapter, we present an approach that clearly divides control over the basic underlying motion (due to inertial and viscous forces) from that of secondary effects of the fluid (affected by surface forces). This makes it intuitive for the animator to fine-tune specific aspects of the animation while meeting the basic design goals for the bulk motion. Our method takes in a dense sequence of *control meshes* as its input. The animator can then run the simulator and adjust some simple dials that select the *drippiness* or *looseness* of the water surrounding the target shape. In addition, the

animator can add specific surface details such as bumps or introduce surface waves to create a more dynamic look while still matching the motion prescribed by the given control meshes.

The key components of our control algorithm are as follows:

- Eulerian control: This allows us to not only reproduce the bulk flow from the keyframes, but also allow for natural inertial effects such as dripping or separating sheets of water from a fast moving shape.

- Mesh-based details: We can generate highly detailed water surfaces and effects (such as ripples and gravity waves) by using a mesh-based representation of the surface that is attracted to the user-provided control meshes.

Unlike previous approaches to this problem, we can control details of the liquid surface at scales smaller than a single grid cell. This freedom allows us to obtain high fidelity effects at reasonable grid resolutions by using a high-resolution mesh to represent the liquid surface while running a simulation on a much coarser grid.

4.2 Algorithm Overview

The input to our fluid control system is a set of per-frame *control meshes* that are provided by the animator. These could either be generated from a sparser set of triangle meshes using the morphing algorithm described in Chapter 3 or could be provided independently by the animator. The only condition imposed on these control meshes is that they must be water-tight and have velocities specified per vertex.

In order to obtain secondary motions, we incorporate these control meshes into a fluid simulator. To compute the Eulerian control forces, the surface of the control mesh for each timestep is rasterized onto the simulation grid and its velocities (computed using finite differences between successive frames) are transferred onto the cell faces of the fluid grid. By setting these velocities as boundary conditions in the

Poisson equation, we can closely track our desired shape or blend it with an unforced velocity field to achieve a looser form of control. We describe the details of this in Section 4.3.

To add surface details, we use a coarse grid-based fluid simulation that is coupled with a Lagrangian surface tracker (as described in [137]). This surface tracker makes use of a *surface mesh* in order to represent the fluid surface. Control forces are computed on both the grid and the surface mesh and this helps us achieve much higher detail than previous methods without incurring a significant computational overhead. The pressure projection step from Section 4.3 produces a divergence-free velocity field. We advect the surface mesh through this divergence-free field and handle all of the topological changes needed to maintain the surface mesh. Finally we compute control forces on the surface mesh vertices to produce a variety of effects ranging from tracking of sub-grid details to the generation of surface waves (see Section 4.4)

4.3 Eulerian Control

The primary reason for using a simulation instead of procedurally animating the provided shapes is to obtain secondary motions that increase the naturalness of the scene. However, any algorithm for fluid control must at the very least, produce results that match the user specified shapes. This notion is not strictly restricted to the geometry of the output but also extends to other aspects of the motion such as the velocity and acceleration. For instance, we would expect to see fast moving water in regions of a rapidly deforming or translating target. This forms the *raison d'etre* for our grid-based control force.

Our Eulerian fluid simulator makes use of a staggered MAC grid, which we shall refer to as the "grid" from this point on. Our simulator broadly consists of the following stages: integration of external forces, advection, and finally, pressure projection.

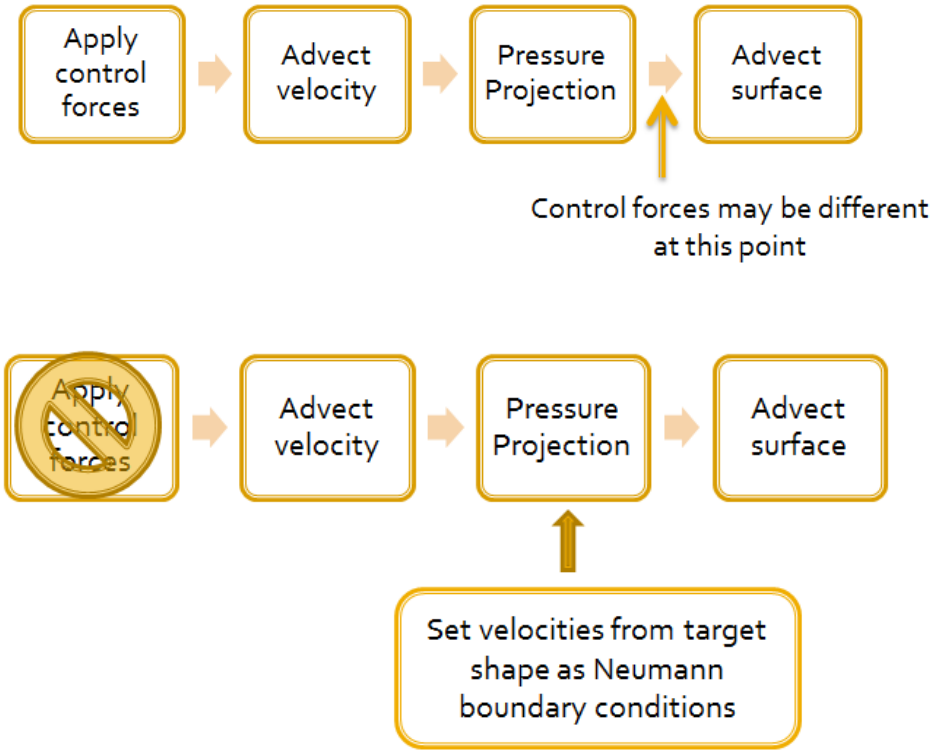

Figure 7: A typical Eulerian fluid simulation pipeline (top) compared against our modified grid-based control algorithm (bottom).

The last step projects the velocities into the closest divergence-free field and enforces incompressibility. We can then advect our current fluid surface through this velocity field to obtain a new surface. While it may be appealing to simply add control forces in the first step of the simulation, this approach can produce unpredictable results because the added forces might be significantly altered by the projection stage. The other concern with this construction is that it also requires us to know the exact control force at each cell in the entire volume of the liquid. This is not a trivial problem because we only have a boundary representation of our target shapes.

Instead, we enforce our control during the projection step by specifying boundary velocities along the control mesh and solve the resulting Poisson equation to obtain a conforming divergence-free velocity field through the entire domain.

This formulation works because the elliptic Poisson equation has the useful property that the solution within an enclosed region is completely determined by the values specified along its boundary. Since our target shapes are meshes that maintain correspondences across frames, we can compute the velocities at the vertices of each control mesh using a simple forward difference. We directly set these as Neumann boundary conditions and obtain new velocities everywhere in the domain that respect these control velocities. The sole restriction here is that the flux through this boundary (i.e. the net amount of fluid flowing through this surface) must be zero in order to preserve volume. If the meshes change in volume, we account for this by modifying the divergence on the right hand side of the Poisson equation.

Figure 8: Tracking targets using only surface waves.

In terms of implementation, we need to rasterize velocities from the control mesh onto the face of the simulation grid. We first rasterize the mesh onto the grid and ensure that the rasterized shape is connected by closing any small air pockets that

might have been formed due to a mismatch in resolutions between the control mesh and the grid. While the velocity field defined on the vertices of the input mesh may have zero divergence, the rasterized field may not obey this property and needs to be corrected. We do this by computing the divergence along the rasterized boundary and distributing it evenly amongst the cells on the surface. We use a standard discretization of the Poisson equation and assign these rasterized boundary velocities as Neumann conditions. We solve the linear system using the preconditioned conjugate gradient method as described in [23].

4.3.1 Relaxed Control

Using the technique outlined above, the motion of the fluid will track the user defined control meshes closely. However, it is often desirable to have a less constrained motion of water that retains more of its inertia or momentum. We call this *relaxed control*. To do this, we perform an additional pressure projection step without enforcing the boundary conditions from the rasterized control mesh. In Equation 14, F_u denotes the force required to project out the divergent component from the intermediate velocity field u^* without any boundary conditions imposed by the control shape. The normal pressure projection required to enforce the rasterized control velocities results in force F_v (see Equation 15). Note that each of these results in a divergence-free velocity field and by linearly blending these, we obtain a new velocity field that also has zero divergence (Equation 16).

$$u^* + F_u = u \tag{14}$$

$$u^* + F_v = v \tag{15}$$

$$\implies u^* + (1-\alpha)\, F_u + \alpha F_v = (1-\alpha)\, u + \alpha v = u_{relaxed} \tag{16}$$

These relaxed control forces may not track the shape perfectly, but they can produce plausible secondary effects such as sheets flying off a fast moving arm or more

subtle sloshing that are consequences of the artist prescribed motion. There exists a distinction between a typical feedback control scheme and our relaxed control method. In the former, animators tune the value of various gains in order to closely match the shape and velocities of the target and the ideal value for each of these is unknown and varies with time due to the dynamic nature of the simulation. However, in our method, the blending coefficient determines the balance between exact tracking and natural motion and this makes for a more intuitive dial for controlling the animation. We demonstrate results of this relaxed control in the dancing sequence of Figure 4.1 and in our videos.

4.3.2 Volume Refilling

When the control is relaxed to a large degree, water tends to spill out from the control shapes and this typically leads to voids inside the target shape that may be undesirable. We detect the loss of volume by supersampling all grid cells inside the rasterized target shape and checking if they are empty. We then modify the divergence of all cells in the interior of the rasterized target to compensate for the lost volume during the pressure projection. This is similar to the approach described in [62] except that we can accurately compute the volume loss because we know the desired volume of the target. This leads to natural refilling of the shape without resorting to stiff spring-like forces.

4.4 Mesh-based Details

The control scheme described thus far operates solely on the grid despite using control meshes as input. If this scheme alone was used to produce controlled fluid effects, it would require a high resolution grid in order to produce detailed fluid surfaces. Recent work by [130; 137] has shown the benefits of using a mesh-based Lagrangian surface tracking technique coupled with a relatively coarse Eulerian simulation and in particular, these methods can produce impressive sub-grid surface details and motions.

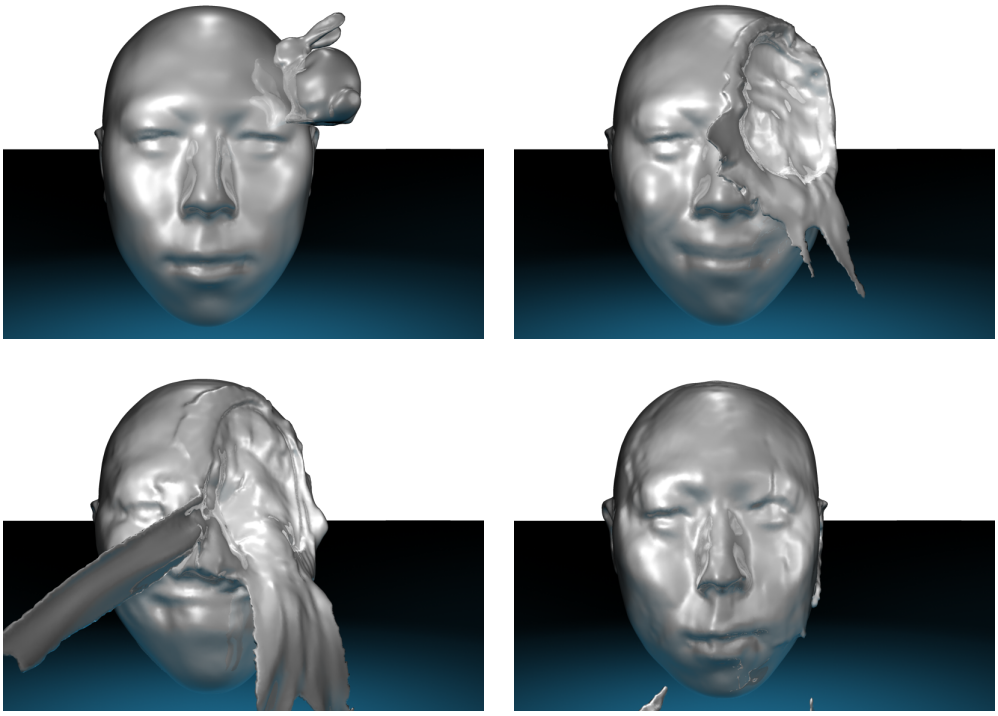

Figure 9: Sequence of images showing the refilling algorithm in action when a fast-moving bunny makes a significant dent in the human head.

In the context of fluid control, such a mesh-based surface representation is attractive because simulation on a low resolution grid is significantly cheaper (the asymptotic complexity of the Eulerian simulation being $O(n^4)$ where n is the grid resolution along one dimension). Consequently, artists can perform multiple iterations to perfect the bulk motion which is imparted by the grid before fine-tuning surface details.

In this section, we describe how we make use of just such a mesh-based surface tracker to give us more fine details and more control of the fluid surface. We will refer to this new mesh as the *surface mesh*, which is advected by the grid velocities and represents the fine details of the fluid surface. This new mesh should not be confused with the user-supplied *control meshes*. We will describe how this surface mesh allows us an additional layer of control above the previously described grid-based method to form fine details such as bumps, spikes and wrinkles, create surface waves and ripples and control the level of drippiness of water around the shape.

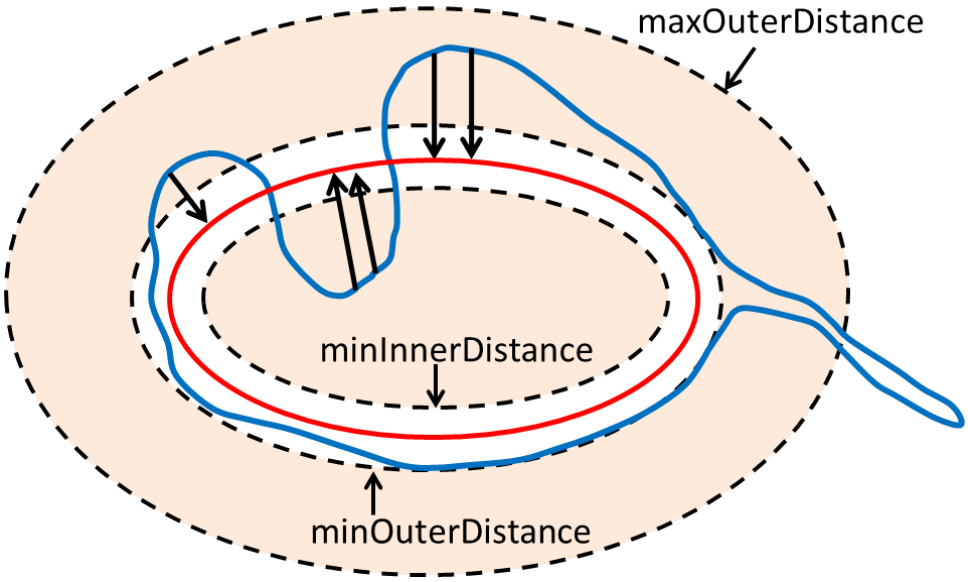

Figure 10: Distance based cutoffs for the mesh attraction force. The user-supplied control mesh is red, and the surface mesh is in blue.

Prior to applying any control force on the surface mesh, we advect the surface through the velocity field produced by the grid based simulation. This velocity field includes the Eulerian control forces that help move the fluid surface towards the target. Next, we handle topological changes by performing operations on the mesh as described in [137].

4.4.1 Capturing Fine Details

The first form of mesh-based control aims to attract the current surface mesh towards the control mesh and match fine details that cannot be captured on the grid. For each vertex of the current surface mesh, we find the closest point on the control mesh. In our implementation, we use a kd-tree as a spatial data structure and insert all points of the control mesh into the tree. We can then query the tree to find the nearest point for a given point on the surface mesh. Next, we apply a positional update using a damped spring equation to pull the surface point closer to the control mesh. The user can adjust the strength of these springs to determine the speed at which the surface is attracted to the control mesh. However, a naive application of such a spring based

force can suppress dynamics near the surface and also prevent the natural separation of water that is far away from the control mesh. We introduce tunable distance based parameters that limit the application of this mesh attraction force (see Figure 10). The control force only acts in the shaded regions and the transitions can be further smoothed by using a linear or quadratic falloff. Using this control technique, we can form sub-grid details such as wrinkles on a forehead, as can be seen in the right image of Figure 14.

4.4.2 Control Over Thin Sheets

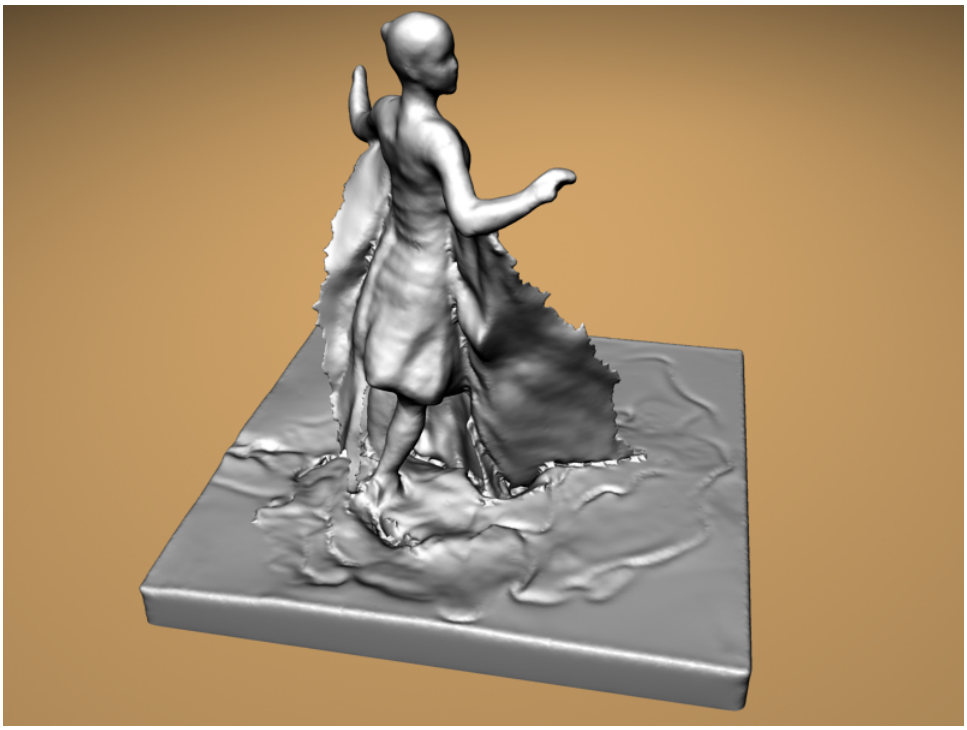

Figure 11: A still from an animation demonstrating the user-controlling tearing of thin sheets. Here, the sheet detaches from the arm as the dancer spins around.

One of the benefits of using a Lagrangian surface tracker is the preservation of thin sheets of water. We defined thin sheets as structures that are nearly flat and that are thinner than a grid cell. Thin sheets do not breakup naturally since they are unaffected by surface tension forces and can persist for long periods of time in the

simulation until they are perturbed due to interaction with other fluid. These sheets can be used to impart a specific look to the animation. For instance, long trailing sheets can indicate high velocities and drippiness, and sheets that quickly break up can be used to preserve detail around important regions of the control mesh.

We can selectively decide to cause a thin sheet to tear, if this is desired for a given animation. To do this, we first identify vertices of the surface mesh that lie on thin sheets as those that lie in a thickened shell that is outside the range of the mesh attraction force and inside a grid cell that is not entirely surrounded by fluid (i.e. at least two of its neighbors along an axis must be air cells). Next, we apply several iterations of smoothing (using mean curvature flow) on only these vertices. This causes a natural tearing of the sheet starting at the minimum distance set by the artist (since vertices below this threshold are being pulled towards the target by the attraction forces). By altering the distance of application and the frequency of this smoothing step, these sheets can be detached at will. For instance, in the dancer animation, we run the thin sheet tearing step once in every five simulation steps. This allows for a more drippy look and creates highly detailed structures that tear off from the arms and torso and are flung away due to their large velocities.

4.4.3 Surface Waves

The previously described control helps the animator track specific details, but does not introduce any additional dynamics on the surface mesh. It is well known that surface waves such as gravity and capillary waves play an important role in creating realistic visual effects at small scales. Capillary waves are a result of surface tension forces that try to minimize the area of the liquid surface while retaining its volume. Due to this, these waves are unlikely to track our control meshes. This inspires us to introduce surface dynamics via waves that are compatible with the underlying animation given by the artist. In order to obtain well behaved waves on our liquid

Figure 12: Tracking targets (top row) using purely surface waves (bottom row).

surface that can co-exist with our other control forces, we need to alter the standard linearized two-dimensional wave equation:

$$\frac{\partial^2 h}{\partial t^2} = c^2 \left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \right) \tag{17}$$

In this equation, c defines the wave speed, while h refers to the current height of the wave at that point on the surface. The right hand side is the curvature of the scalar field h, while the left hand side of the equation is the acceleration at this point which is proportional to this curvature. Intuitively, curvature can be viewed as a measure of the difference of the scalar value at the current point compared to the scalar values at neighboring points. If we set h to be equal to the distance from a point on the surface mesh to the closest point on the control mesh, then the acceleration experienced by that point on the surface will be determined by its displacement from the control mesh relative to its neighbors. To illustrate the effects of this equation,

consider a water surface that is parallel to the control mesh. The points on this surface will not experience any wave forces because they are equidistant from the control shape. Now, if the control mesh were to develop a slight furrow, this would change the relative distances and spawn a wave that will eventually settle down once the sheet takes the shape of the control surface.

Note that this sheet does not need to exactly settle down on the control mesh for it to not experience any force - a parallel displacement will also result in zero curvature. However, we can limit the magnitude of such a displacement by using the mesh attraction forces described in the previous section. Specifically, by tuning the *minInnerDistance* and *minOuterDistance* (see Figure 10), we can determine how far surface may move away from the control mesh before being pulled back. These vertices that move beyond the inner distance thresholds are damped and act as a damped boundary conditions in the wave equation solve. Further, if certain vertices have a high velocity component that is normal to the surface, they can quickly move beyond the influence of both wave and attraction forces. Hence, this setup does not preclude the possibility of liquids sheets detaching themselves from the main body of water and allows for waves to co-exist with our other control techniques.

We use the approach defined in [130] and solve this equation using the Implicit Newmark time integration scheme. This amounts to solve the following sparse linear system (where $\mathbf{h}$ is a vector of the heights of all surface points, and L is the matrix representation of the Laplacian operator on $\mathbf{h}$):

$$\left(I - \frac{\Delta t^2}{4} c^2 L\right) \mathbf{h}^{n+1} = \mathbf{h}^n + \Delta t \mathbf{h}_t^n + \frac{\Delta t^2}{4} \mathbf{h}_{tt}^n \tag{18}$$

We update the vertex positions of the surface mesh by displacing them along their normals by the new height values obtained by solving the above equation. We do not include vertices that are beyond the inner distance thresholds while solving this equation.

In some cases it may be desirable to prevent waves in user defined regions to preserve features. We allow the user to specify vertices that must be preserved exactly by painting weights on the mesh. A weight of 1 implies that the vertex is unaffected by the wave equation and is pulled towards the control mesh using the mesh attraction force. A weight of 0 indicates that the point is moved solely by the wave equation. We then run the diffusion equation on the mesh for several iterations to spread these weights. The final displacement of a vertex is a linear blend (using this diffused weight) between the position predicted by the wave equation and the position determined by the attractive forces. This approximates non-reflecting boundary conditions reasonably and preserves details where desired. We used this approach to preserve the details of the mouth and eyes for the animation of Figure 8.

4.5 Results

All of our fluid simulations were run on 8 cores of a dual processor Intel Xeon with 2.4 GHz CPU's and 6 GB of memory. Our control scheme typically adds an overhead of 15% - 30% to the computational cost of a regular fluid simulation. Most of the sequences averaged under 10 seconds per frame at a grid resolution of 100^3.

See Table 2 for grid resolutions, surface mesh triangle counts and fluid simulation times. Note that all of our simulations ran in under an hour.

The liquid bunny sequence of Figure 13 demonstrates our volume preserving morpher. In this sequence, a bunny is pulled out of a pool of water, it is deformed by twisting, stretching, and squashing, and then it is dropped back into the pool. For

Table 2: Grid resolution, surface mesh triangle counts (from a typical frame), and simulation times (in minutes) for our animations.

Sequence	Frames	Grid res.	Triangles	Sim Time
Dancer (lo)	500	32^3	70,502	18
Dancer (hi)	500	64^3	207,886	56
Drippy head	200	50^3	136,596	22
Bunny Bending	275	50^3	259,250	29

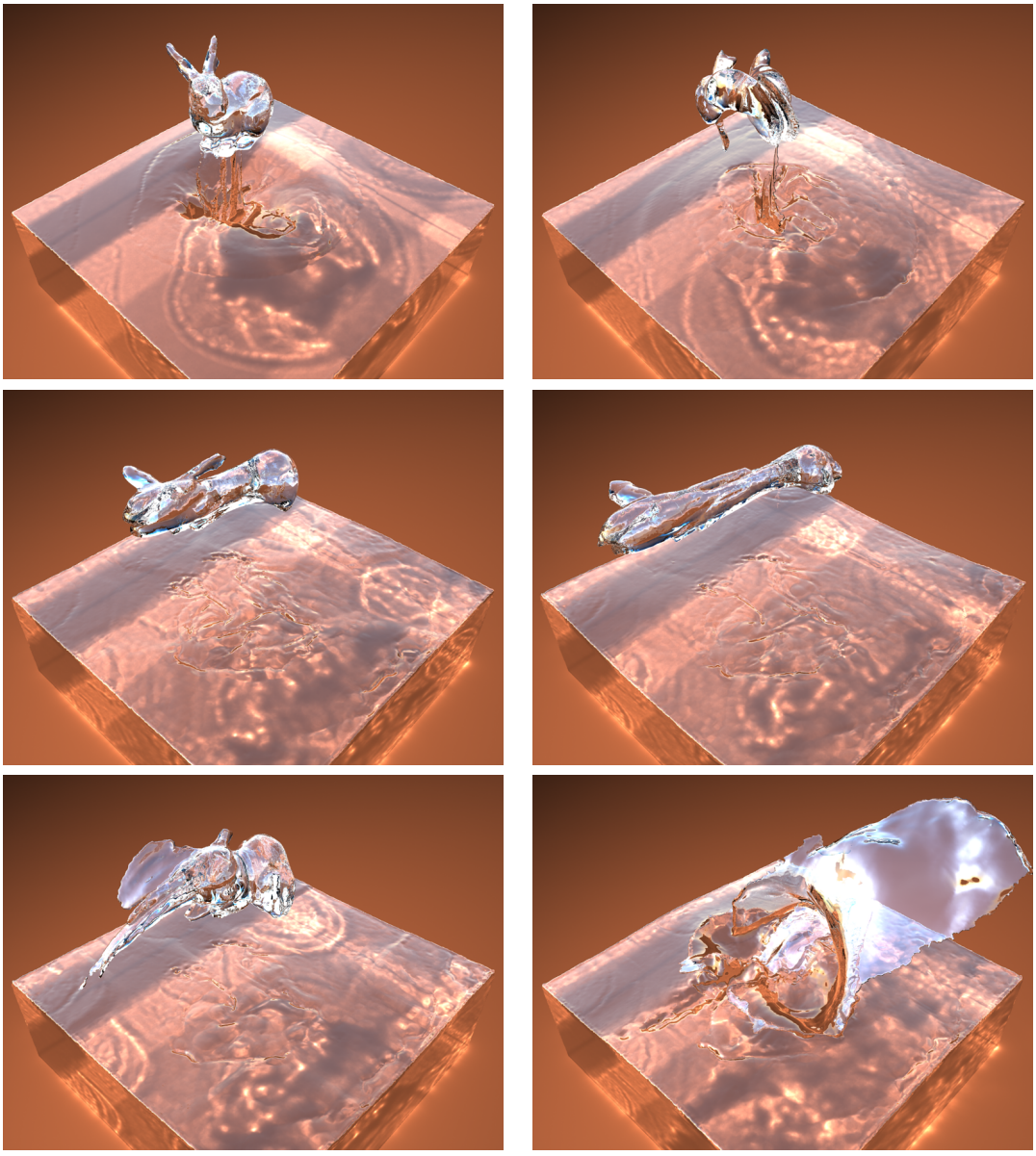

Figure 13: Water bending: A bunny is pulled out of a pool of water, twisted, stretched, squashed, and is then dropped back into the pool. The control meshes for this sequence were sparse, and the per-frame control meshes were generated using our volume-preserving morpher.

this sequence, we provided only a coarse sequence of control meshes for the various bunny shapes. Our volume preserving morpher then created the per-frame control meshes that were used for the sequence. The relaxed control parameter was set to $\alpha = 0.5$ for this scene in order to allow water to easily drip out of the deforming

bunny. The pool of water in this example is not controlled at all, yet it interacts gracefully with the moving bunny.

In our video, we show a breakdown of some of the elements of the morphing bunny sequence. We first go through the morphs, without any fluid simulation. Note that we added velocity constraints to the stretch and squash sequences. This is especially important at the end of the squash sequence when we release control of the fluid and the momentum of the fluid creates a large splash. We also show examples of using different α (blending) values with the bunny, showing various degrees of relaxed control.

Figure 4.1 shows our water dancer example. The control meshes for this sequence were already given per-frame, so our morpher was not used. These meshes are from the multi-view silhouette approach to mesh animation by Vlasic et al. [133]. Note that these input meshes do not conserve volume through the course of the animation, but our divergence correction managed to deal with this without introducing noticeable artifacts. This sequence, like that of the bunny, shows that the fluid closely tracks the control mesh, yet fluid is still allowed to escape in natural drips and sheets. This sequence also demonstrates the benefit of attracting the fluid surface to the control mesh. Even though this simulation was run at a resolution of 100^3, the attraction of the fluid surface to the control mesh preserves many of the fine details of the control mesh.

In the accompanying video, we demonstrate surface waves in several variations of the head animation sequence shown in Figure 14. Like the dancer sequence, the control mesh for this sequence was provided on a per-frame basis, and in this case the meshes were produced by the single-view reconstruction method of Li et al. [78]. Our first sequence of the head shows an expression change from smile to frown, in zero gravity. The motion of the jaw produces ripples on the cheeks and across other portions of the face, and these ripples are purely due to our solution of the wave

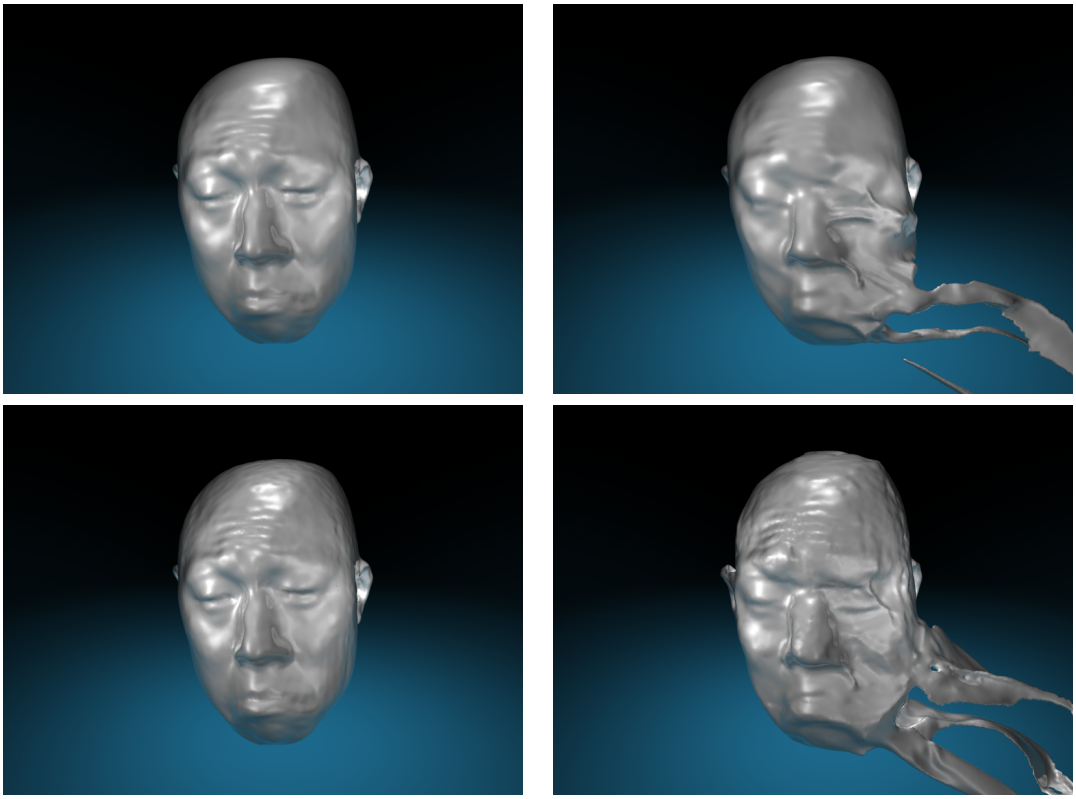

Figure 14: Different effects can be achieved by varying the control parameters: with and without waves (bottom and top rows respectively), tight and loose tracking (left and right columns respectively).

equation on the fluid surface mesh. This demonstrates that our control waves can track targets without resorting to any grid based force. Other sequences include gravity and use an animation of the head shaking back and forth. We demonstrate the effects of relaxed control (using a blend factor of 0.25) as well as those of mesh-based attraction and surface waves in Figure 14. A variety of behaviours ranging from tight tracking to drippy shapes with dynamic surface waves can be achieved by using a combination of these control parameters. Finally, we demonstrate how our control technique responds to external forces when a fast moving bunny collides with the animated head. In this example, we did not enable waves and used a blend factor of 0.2 for the relaxed control.

4.6 Discussion

We believe that the approach presented in this chapter is best suited to animations of shapes or characters that can be fully prescribed by the animator. While it could be used for more natural animations, the challenge lies in providing the appropriate inputs (watertight meshes of a detailed liquid surface) and this might prove to be time-consuming for an animator. Compared to earlier approaches to liquid control [112; 104], our method exposes controls that can alter both inertial and surface behaviour of a liquid while still tracking the input meshes. This control over the output does not impose a large computational overhead unlike the method proposed by McNamara et al. [83]. We think that the collection of algorithms presented in this chapter makes for a practical animation system.

There are a few limitations to our mesh-based approach to fluid control. If our volume preserving morpher is to be used, the input meshes must have the same vertex connectivity. We note that there are several published approaches for taking two existing meshes and creating vertex correspondences between them [73; 69; 110]. If our morpher is not required, then each control mesh can have a distinct connectivity from the others. In this case, we would also require per-vertex velocities to be provided.

A second limitation of our approach is that our control mesh sequences cannot move too fast. High velocities do not automatically destabilize the fluid simulation, but they can cause abrupt changes in wave heights as well as topological changes that may manifest themselves as surface artifacts or aliasing. The solution is to take smaller timesteps and linearly interpolate between the keyframes to obtain additional inbetween frames. This comes at the cost of introducing artifacts due to the interpolation.

We must also point out that there is a tradeoff that needs to be made when picking the resolutions of the surface mesh and the simulation grid that it is embedded in. At low grid resolutions, surface features that are smaller than a grid cell in size will persist until they are displaced by the motion of the liquid or resampled due to topological events. This can impart a viscous appearance to the liquid. To avoid this, we recommend that the animator selects a grid resolution that captures the desired dynamics and then picks a surface resolution whose average edge length is roughly a quarter of that of a grid cell.

4.7 Future work

We have presented a system for controlling the motion of fluids, with a particular emphasis on creating and tracking specific shapes. The unifying theme of our approach is the control mesh that the animator provides in order to guide the fluid motion. If only a sparse set of such control meshes are given, then our system uses a volume preserving morph to provide a dense set of meshes. For a given time step, the control mesh is rasterized onto a grid, and this rasterization provides velocities that act as boundary conditions during the pressure projection step of fluid simulation. The fluid can strictly follow the control mesh, or the control can be more relaxed if the animator chooses. Finally, surface waves can be generated by attracting the fluid surface to the control mesh and also by solving the wave equation on the fluid surface.

There are several avenues for future work. One possibility is to use our control techniques for fluid simulation methods other than Eulerian grids, such as those methods that represent the fluid as tetrahedral elements. Also, since we have an explicit mesh for tracking the fluid surface, we could use this mesh to carry along

foam and other materials on the fluid surface. Finally, our examples demonstrate the creation of moving 3D shapes in fluids, which can be used for film effects. We think that our method can also be used to create more subtle fluid control effects such as changing the shape of a breaking wave, however providing realistic looking input shapes could prove to be extremely time-consuming for the animator.

CHAPTER V

SPACETIME SURFACES

Animation has traditionally been concerned with producing the illusion of smooth motion from a set of discrete, static images. Manual devices like phenakistoscopes, zoetropes and flip books required the artist to draw a set of images that were then displayed in rapid succession. This idea has carried over to computer-based animation where the animator digitally sketches out or sets the pose of a character using 3D rigging software at discrete instants of time. We can treat each frame of animation as an independent entity (such as a 2D sketch, a triangle mesh or a tetrahedral volume) that is fully defined at a specific instant of time. This representation has the advantage of being simple and extremely flexible in that no additional information is required beyond the spatial data from the frame in question. For discretized representations such as triangle meshes, there are no additional requirements imposed on their connectivity or topology.

However, for animation data that is a result of a time-evolving process such as a physical simulation, successive frames are clearly linked by the velocity field. For instance, consider an animation that consists of frames of a water surface. The surface at time $T + 1$ is produced by advecting the surface at time T through the velocity field in the simulation. When we consider this link information along time for each vertex, we have a higher dimensional representation of our animation which is now embedded in spacetime. There are several reasons for picking this representation:

- Continuity: We can treat each animation as a manifold with boundary in spacetime and hence as continuous data. Note that not all points may have a future neighbour in time due to changes in topology of the surface. For instance, the vertices at the bottom of each of the drops in the sequence shown in Figure 5.3 get deleted when they merge with the pool. Nevertheless, the spacetime mesh can be viewed as a piecewise linear approximation of the liquid surface over time. As a consequence, any smooth deformation of this surface will yield an animation that will be temporally coherent.

- Intra-frame temporal variation: Once the individual frames of animation have been converted into a spacetime mesh, the restriction that all vertices from a single frame must lie on the same time hyperplane is removed. This permits local deformations along time and allows us to model behaviours that would be impossible for algorithms that preserve the frame structure (only global retiming).

- Handling of topological changes: One of the challenges of object registration involves dealing with meshes that have different topologies. In particular, liquid simulations are characterized by numerous changes in the topology of their surfaces. With our spacetime representation, changes such as merging or splitting of water bodies do not need special treatment since the spacetime surface remains a single connected component.

- Sparsity: Unlike volumetric approaches, the spacetime mesh is a compact representation that can be represented with a sparse connectivity graph.

5.1 Definition

We define a spacetime mesh as a smooth manifold with boundary in $\mathbb{R}^n$ that consists of a set of vertices $V = \{\mathbf{v_i}\}$ and a set of edges $E = \{\mathbf{e_i}\}$. Each $\mathbf{v_i} = (t_i, x_{i,0}, x_{i,1}, ..., x_{i,n-1})$ where t_i is the time coordinate of the vertex, $x_{i,0}, x_{i,1}, ..., x_{i,n-1}$ are the spatial coordinates of the vertex i expressed in the canonical basis. An edge $\mathbf{e_i} = (j, k)$ exists if $\mathbf{v_j}$ and $\mathbf{v_k}$ are lie in a neighbourhood of each other.

5.2 Properties

We begin by listing a set of properties that we expect the spacetime mesh to satisfy:

1. Each vertex $\mathbf{v_i}$ must have at least one neighbour in time.
2. It should be orientable. In other words, given a point $\mathbf{p}$ that lies on the surface, we must be able to make a consistent choice for the normal to the spacetime surface at that point.
3. Given a point $\mathbf{p}$ in $\mathbb{R}^n$, we must be able to find the closest point on the spacetime mesh (in the L2 sense).

Each of these properties is essential for performing the registration of two spacetime meshes. Property 1 avoids the degenerate cases where a point exists exactly for one frame. It also allows us to define a unique surface normal at each point. Property 2 enables a number of useful operations including correspondence assignment based on compatible normal orientation and surface displacements. Property 3 ensures that

we can compute a set of approximate correspondences between two meshes using the Euclidean metric to measure proximity.

5.3 Construction

For illustrative purposes, we choose to depict a 3D spacetime surface constructed from a 2D animation. In the discussion of the construction, we will assume that our animation is in $\mathbb{R}^3$ and consists of a set of 3D triangulated meshes. We will consider the construction of spacetime meshes from other representations such as level sets later in this dissertation. .

To construct a space-time surface, we first represent each vertex $\boldsymbol{v}$ at time t as $(t{\cdot}e, v_x, v_y, v_z) \in \mathbb{R}^4$ where e is the average length of all edges in the first frame of the animation sequence. This ensures that space and time dimensions are scaled uniformly. We then wish to connect the individual meshes in time. Instead of using a generic animation reconstruction algorithm [127], we can take advantage of the information embedded in our fluid simulations. We advect each vertex $\boldsymbol{v}_i^t$ using the velocity field $\boldsymbol{u}_i^t$ for an interval Δt to its predicted position $\widetilde{\boldsymbol{v}}_i^t$. We then find the vertex $\boldsymbol{v}_j^{t+1}$ on frame $t{+}1$ that is closest to $\widetilde{\boldsymbol{v}}_i^t$. If the geodesic distance between the two is less than twice the average edge length ($2e$), then we create a link between $\boldsymbol{v}_i^t$ and $\boldsymbol{v}_j^{t+1}$. This link will not exist when a vertex is deleted during a topology change. For instance, two surfaces will collide and disappear where they merge together; a vertex on one of these colliding surfaces will have no corresponding vertex in the next frame.

With this time link information, we can compute normals by creating a vector

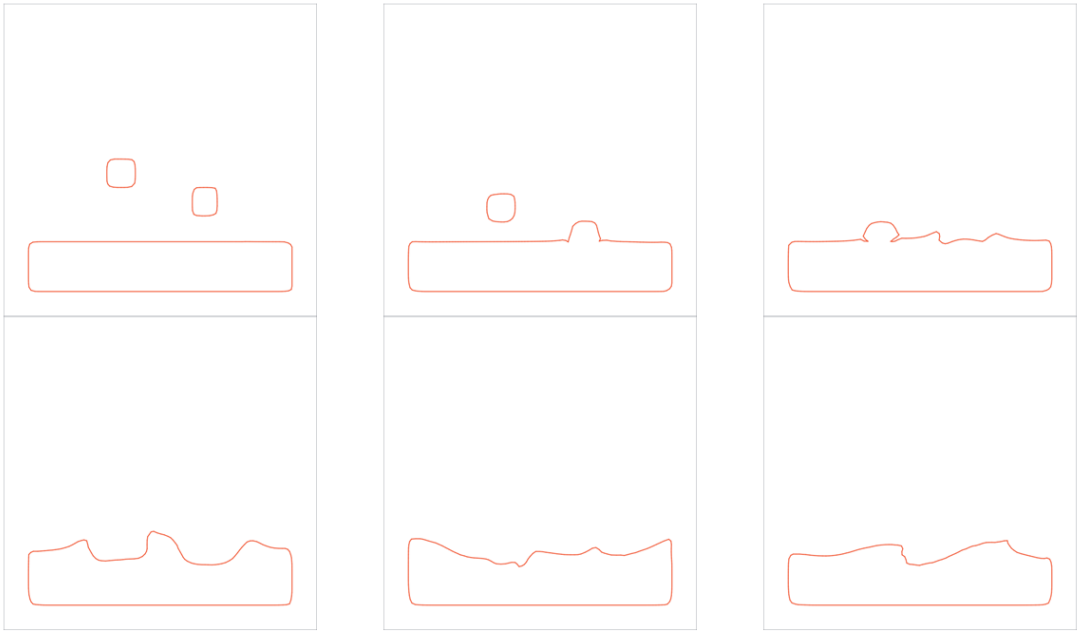

Figure 15: A set of frames from a 2D animation.

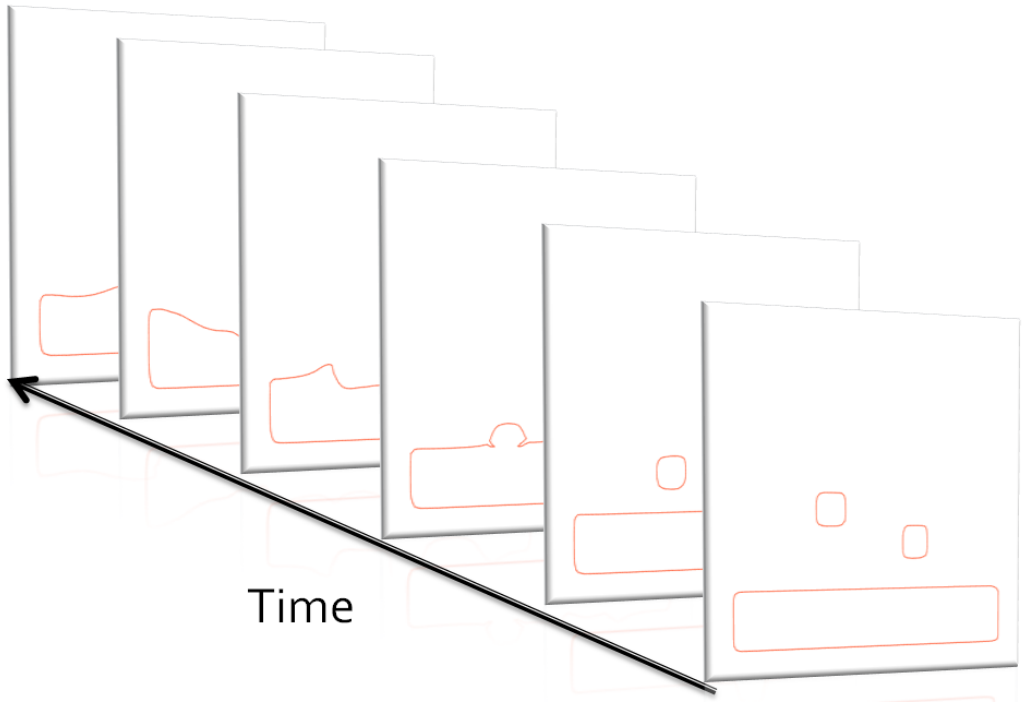

Figure 16: 2D frames are stacked up in time.

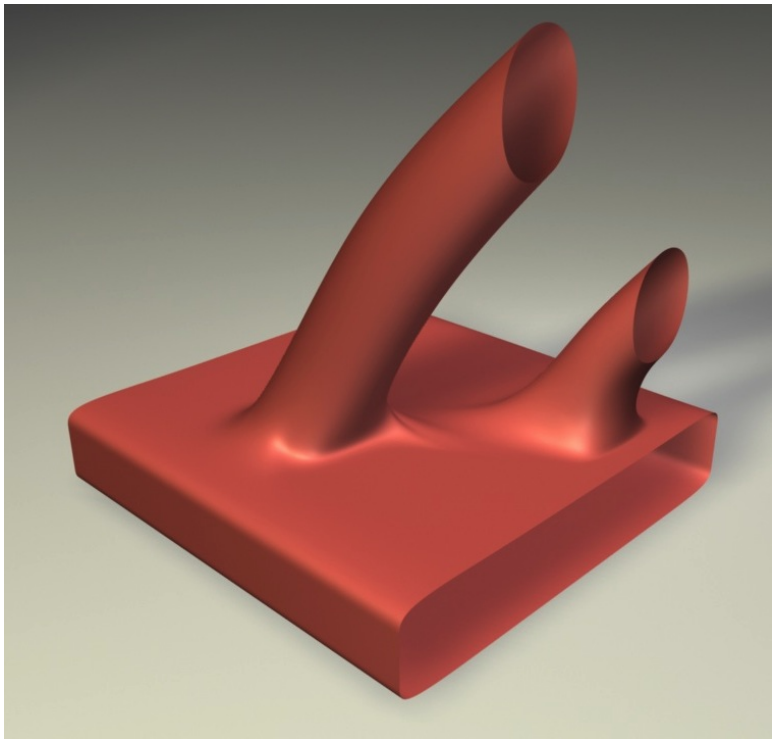

Figure 17: An illustration of a 3D spacetime surface created from the frames of a 2D animation of two drops falling into a pool of water.

that is orthogonal to the tangents in space and in time; we now have an oriented surface corresponding to an animation. Note that this representation is not a fully tessellated mesh in spacetime, but instead is a graph where each vertex has a set of neighbors in space and up to two neighbors in time. In other words, this graph provides connectivity information as well as orientation at each vertex by stacking frames next to each other in time and loosely connecting them with links that are guided by the velocity field. When required, we can construct a local approximation of the spacetime surface near each vertex on the fly as described in Section 6.3.5.

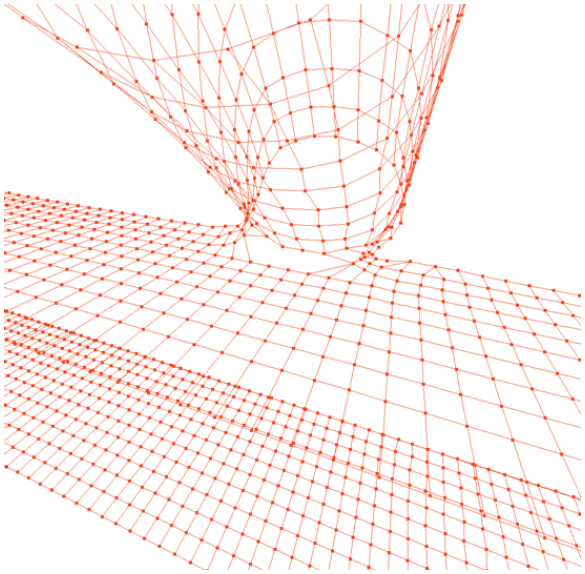

Figure 18: A 3D spacetime mesh with boundary.

5.3.1 Normals in spacetime

Note that we cannot simply compute the normal using a cross product of vectors in the tangent plane as we would in the case of 2D animations since this notion does not exist in $\mathbb{R}^4$. Instead, we will use exterior calculus to compute the hodge star of a wedge product of vectors in the tangent space of the point on our 4D manifold.

In $\mathbb{R}^n$, we have:

$$*\left(\boldsymbol{e_0} \wedge \boldsymbol{e_1} \wedge ... \wedge \hat{\boldsymbol{e_j}} \wedge ... \wedge \boldsymbol{e_n}\right) = (-1)^{j-1}\, \boldsymbol{e_j} \tag{19}$$

where $*$ is the hodge star operator, $\boldsymbol{e_i}$ represent basis vectors in $\mathbb{R}^n$ and $\hat{\boldsymbol{e_j}}$ indicates that the vector is excluded from the wedge product. This means that given $n-1$ linearly independent vectors expressed in the natural basis, we can compute a vector that is orthogonal to each of them. Applying this to $\mathbb{R}^4$, given three tangent vectors

$\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$, we obtain the equivalent determinant expression that gives us the desired normal in the standard basis when expanded using the first row:

$$\boldsymbol{n} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ u_0 & u_1 & u_2 & u_3 \\ v_0 & v_1 & v_2 & v_3 \\ w_0 & w_1 & w_2 & w_3 \end{vmatrix} \tag{20}$$

We will still need to ensure that normals are oriented consistently because interchanging two vectors (rows) will flip the sign of the determinant. There are two possible solutions to this. The simplest method would be to have a consistent ordering of edges that are incident on a vertex. This can be easily achieved using a corner table based implementation. The other option is to pick an arbitrary vertex, compute the normal and walk along the mesh, adjusting the sign for each normal $\boldsymbol{n_i}$ at each vertex $\boldsymbol{v_i}$ on our manifold so that their signs agree.

5.3.2 Surface representation using local tetrahedralization

So far, our discussion of the spacetime mesh has focused on the connectivity of the vertices. In order to compute quantities at other points on the surface, we first need to establish a clear notion of the surface. Our input animation consists of triangulated meshes in 3D. We would like our constructed surface to retain these triangular faces. One way to do this would be to perform a constrained global tetrahedralization in 4D such that the surface includes all the faces from the individual meshes. However, this is a very challenging problem that is likely to be computationally expensive to solve. Instead, we opt for an approximation of our surface by creating a local

tetrahedralization. The purpose of this tetrahedralization is to "fill in" the regions between pairs of meshes that come from the different discrete frame times. We must stress that this tetrahedralization is entirely different from filling the volume of the triangular mesh (that comes from each frame) with tetrahedra. In constrast, we are creating a 3D shell around a 4D spacetime volume and we describe the process in detail below.

First, we consider every face on the triangle mesh at time T and following the time links at each of its vertices to obtain corresponding positions at time $T+1$. This yields a triangular prism in 4D for each face at time T. In order to tetrahedralize these prisms, we require that adjacent prisms use the same diagonal edge to split their shared quadrilateral face. That is, the diagonal assignments must be consistent between adjacent prisms. This choice is important because the quadrilaterals are not always planar.

There are two distinct ways to assign diagonals to a single prism. The Type 1 prism can easily be split into just three tetrahedra (see Figure 19, upper left). Unfortunately, the best dissection of the Type 2 prism (Figure 19, upper right) that we have found leads to eight tetrahedra. We form these eight tetrahedra by introducing a new vertex at the center of the prism, and then creating one tetrahedron per triangular face by connecting it to the new center vertex. Since we want to avoid creating more tetrahedra than necessary, we desire an assignment of diagonals to the prisms that includes as many Type 1 prisms as possible, and few or no Type 2 prisms.

We have devised a simple algorithm that creates a consistent assignment of diagonal edges across all of the prisms, and that also creates only Type 1 prisms. To do this, we only have to consider the triangle mesh from one of the two frames that are to

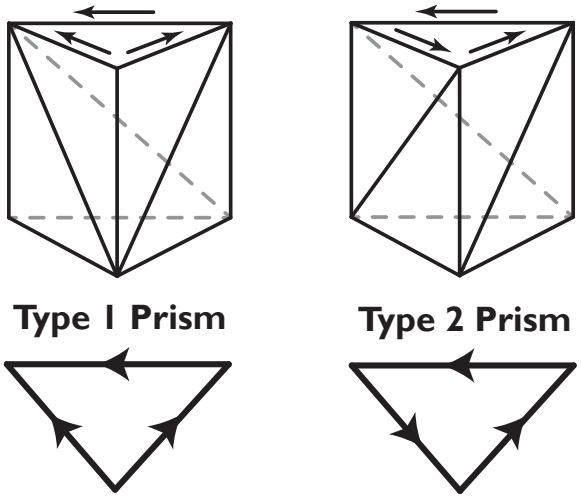

Figure 19: The two types of triangular prisms, based on the orientation of the diagonals that split their quadrilateral faces. Type 1 prisms (top left) can be divided into three tetrahedra, but Type 2 prisms (top right) require eight. The two triangles (bottom) show the directed edge labels that correspond to the two types of prisms.

be connected by prisms. We will label each of the edges of this triangle mesh with a directed edge. Each of these directed edges indicates the orientation of a diagonal for the corresponding prism quadrilateral face. Specifically, a directed edge that points towards a particular vertex $\boldsymbol{v}$ indicates that the diagonal edge for that quadrilateral has $\boldsymbol{v}$ as one of its vertices. A triangle that is labeled with edges that are all oriented either clockwise or counter-clockwise (Figure 19, lower right) indicates a Type 2 prism, which is to be avoided. A triangle with mixed orientations indicates a Type 1 prism (Figure 19, lower left), which is desired. Our diagonal assignment algorithm operates purely on the directed edges of the triangle mesh.

To generate the edge directions for a given triangle mesh, we start by assigning a unique numerical label to each vertex of the mesh. Then, each edge of the mesh is assigned the direction that points from the vertex with the lower value towards the vertex with the higher value. This simple rule gives a direction to each edge in the mesh. Note that for any triangle, one of its vertices always has the highest

numeric label among the three vertices. This means that two of the directed edges will point towards this vertex, indicating a Type 1 prism (Figure 19, lower left). Since it is not possible for the three vertex indices to form a closed cycle of numeric labels (e.g. $v_1 < v_2 < v_3 < v_1$), a Type 2 configuration (Figure 19, lower right) is impossible. Since we globally assign the numerical vertex labels, adjacent triangles share the same understanding about the direction of their shared edge. This means our diagonal assignment is consistent between adjacent prisms.

Once we have this local tetrahedralization, we can compute the normal at any point on the surface. We first find the tetrahedron that contains it and perform a barycentric interpolation of the the normals at the vertices.

5.3.3 Computing the closest point

Most registration algorithms require the ability to compute the closest point $\mathbf{c}$ on a surface to a given point $\mathbf{p}$. We assume that our surface is sampled uniformly and that the closest point lies in close proximity to the closest vertex on the surface. We create a kd-tree with all the vertices of our spacetime mesh and query it to find the closest vertex on the surface.

This vertex has a set of incident triangles (from the triangulated mesh corresponding to that frame). Each of these triangles has an associated triangular prism in spacetime, which in turn has three tetrahedra that are created according to the discussion in the previous section. The closest point lies inside one of these tetrahedra. We also know that the closest point satisfies the property that:

$$(\mathbf{c} - \mathbf{p}) \cdot \mathbf{n} = 0 \tag{21}$$

We find this closest point for each tetrahedron, and pick the one that has the least distance to $\mathbf{p}$ and also lies inside the tetrahedron.

5.4 Surface extraction

In this chapter, we have placed much emphasis on the construction and properties of spacetime meshes. Once we have performed our desired operations on a 4D spacetime mesh, it is important to be able to retrieve 3D data that we can visualize. Since an animation typically has a constant frame rate, we need to retrieve 3D meshes from the 4D mesh by intersecting it with regularly spaced time hyperplanes.

We first construct a local explicit tetrahedralization of the entire space-time mesh as described in Section 5.3.2. We then slice this tetrahedral mesh with hyperplanes of constant time, and each such slice yields a triangle mesh for a given frame time. In order to make this process more efficient, we make a couple of optimizations.

First, we use bounding boxes to exclude the majority of the spacetime mesh from the geometric intersection tests. This is necessary because triangles from a frame of an animation that has been deformed do not always have the same time coordinate. In other words, frames can be curved along time. We create an axis aligned bounding box in spacetime per frame by finding the minimum and maximum of each coordinate of the vertices of the triangular prisms associated with its faces. We now only need to consider frames whose bounding boxes are intersected by the time hyperplane under

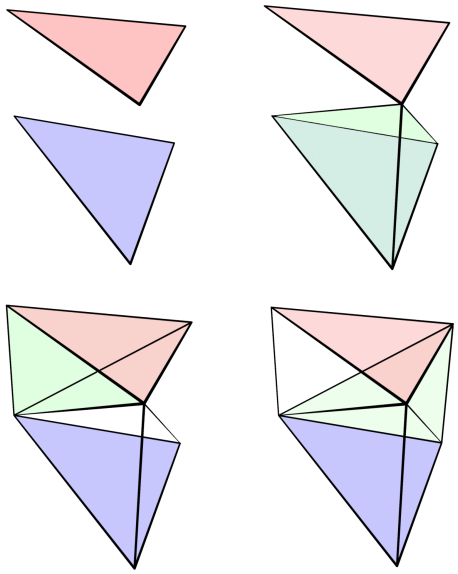

Figure 20: Splitting up of a triangular prism into tetrahedra.

consideration.

Next, we treat each triangular prism independently due to our consistent treatment of the diagonals and as a consequence, we are able to process these in parallel. Note that each triangle from a frame yields exactly one triangular prism that connects it to the corresponding triangle on the next frame. Because our surface tracker attempts to maintain the surface triangulation between frames of the animation, almost every triangle from one frame has a matching triangle in the next. In these cases, the corresponding prisms are well defined. In the cases where our tracker performs mesh clean-up operations such as edge splits or edge collapses, we can still create a small group of tetrahedra to fill between the time slices. We break each prism down into three tetrahedra and use fast triangle-plane intersection tests against the faces of each tetrahedron. The intersection of a tetrahedron with a plane yields either a triangle or a quadrilateral which is then split into two triangles. Due to this splitting of a prism into tetrahedra, the 3D meshes extracted from a spacetime mesh typically

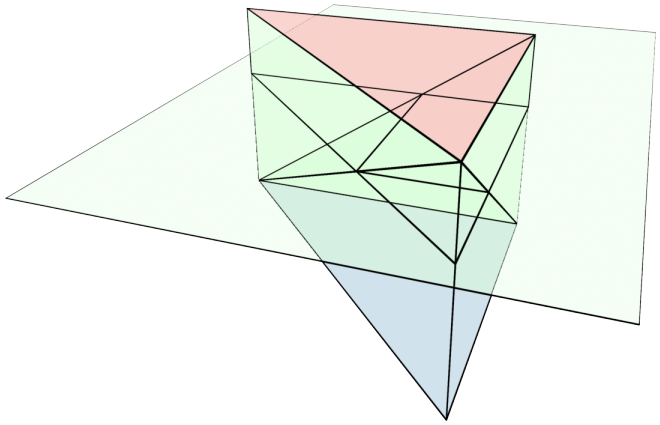

Figure 21: Intersection of a time hyperplane with a triangular prism that has been tetrahedralized.

have more triangles than the source animation as can be seen in Figure 21. It might be possible to perform a simplification step per frame if there are strict requirements on triangle count or size.

5.4.1 Hole Filling

The algorithm described thus far can produce meshes that are not water-tight. This happens due to a couple of reasons. First, whenever there is a resampling event or a topological change such that the connectivity of a mesh changes, there is a possibility that all vertices of a triangle from frame T do not exist in frame $T+1$. Further, even if they happened to exist, there is no guarantee that these vertices form a triangle in frame $T+1$. In such scenarios, we simple avoid a local tetrahedralization and resort to a hole-filling algorithm that sews up any open holes in the extracted mesh.

There have been numerous methods proposed in graphics literature to tackle this problem. However, in our case, the resulting holes tend to be quite small relative to the size of the mesh and can be filled with a simple approach. We fill these holes by introducing a new vertex at the centroid of the hole vertices and constructing a triangle fan that radiates from the centroid to all of the boundary edges. This method worked very well on our test cases and we did not see the need to replace it with a more sophisticated technique. However, if a hole happens to be highly non-planar, then there is a possibility of this algorithm producing self-intersecting geometry.

5.5 Discussion and Future Work

5.5.1 Extension to other fluid representations

All the results related to spacetime meshes in this dissertations were generated using inputs from a mesh-based fluid surface tracker. A natural question at this point is whether we can construct spacetime meshes from other fluid representations such as level sets or particle-based fluids. We discuss possible approaches and outline the likely challenges in this section.

The first thing we need to do is to identify the vertices of the spacetime mesh. Typically, particle-based fluids also resort to using iso-surfaces of a scalar field in order to define their surface. As a consequence, we believe that a unified treatment of both level sets and particle-based fluids is possible. We can sample the surface by either shooting rays through the scalar field at each frame or by running a piecewise linear iso-surface extraction such as marching cubes and sampling the resulting geometry. We can then associate spatial neighbours to each vertex by picking a small number

of points that are nearby and have compatible normal orientations.

Next, we need to ensure that we can satisfy the properties required from a space-time surface:

- We can find a neighbour in time by tracing through the velocity field and finding the closest point to the advected point using a kd-tree built from the sampled points. We could be more precise by moving along the gradient of the signed distance field of the next frame until we cross the zero iso-contour.
- We can compute a normal using the method described in Section 5.3.1 since we now have neighbours in space and time.
- We can find the closest point on the mesh using either the signed distance field or a kd-tree as indicated earlier.

It is clear that we can build well-defined spacetime surfaces from some common representations of fluids. The real challenge lies in the surface extraction step. We see a couple of possibilities here. One option might be to perform an intersection of a time hyperplane against all links that connect two vertices along time and space. Once we have a set of intersection points, we can reconstruct a surface from them using a method like the algebraic point set surface reconstruction [48].

The other option is to convert each of these representations into a triangle mesh at each frame (by using an iso-surface extraction algorithm such as marching cubes or marching tetrahedra) and use the mesh-based construction described in this chapter. The main difference with a mesh-based surface tracker is that we are not guaranteed

any temporal coherence in terms of mesh connectivity and this is likely to lead to more holes in the surface that will need to be filled as a post-process.

5.5.2 Application to other phenomena

We have used spacetime meshes primarily to represent liquid animations but it is possible that they could be used with other animated phenomena such as cloth and deformable bodies. Liquid surfaces are more difficult to handle because they are subject to numerous and frequent topological changes. This is unlikely to be the case with cloth where the mesh retains its connectivity through the course of the animation. The main question then is to determine whether the applications of a spacetime mesh serve these other phenomena as well as they do liquids.

5.5.3 Memory footprint and compression

A concern with the spacetime mesh representation is that it increases the dimensionality of the data and inflates the size of an animation. As a result, long sequences with high resolution meshes can take up on the order of gigabytes of memory. There has been active research on the compression of meshes in memory such as [49] and these might offer a good direction for future work. It is likely that most applications will require queries to various parts of the mesh in a non-sequential order, so it would be important to choose a compression scheme that has an efficient random access operation. Another possible improvement might lie in the area of multi-resolution spacetime meshes where a coarser representation could be used in place of a detailed mesh.

In the next chapters, we will look at how we can use these spacetime surfaces to blend between animations.

CHAPTER VI

BLENDING LIQUIDS

6.1 Introduction

The ability to direct and fine-tune visual effects is one of the main reasons for their popularity. This is especially true for effects that are as difficult to control as fluids. In Chapter 4, we presented an algorithm that enables animators to create a liquid animation that tracks a set of meshes. However, this workflow is not ideal when applied to more realistic scenarios such as flowing water or splashes where the animator would have to spend a great deal of effort to craft these keyframe meshes. As a result, we need to consider alternative means of dealing with these kinds of effects.

A common approach in visual effects production is to run simulations in batches to explore the tuneable parameter space (for instance, the initial velocity of a moving object, gravity, viscosity etc.). Afterward, the artist and supervisor can select the simulation that best matches the desired goal. If none of the outputs is chosen, the parameter range is narrowed down to run additional simulations. This workflow removes the need for keyframing or external forcing of the simulation and redirects the effort into finding the appropriate setup and initial conditions for the desired shot.

However, there are several downsides to this approach: the new simulations can result in undesired behavior (like a distracting splash in the center of attention), it

may require an excessive number of iterations of expensive fluid simulations, and the data from early simulations are wastefully discarded. This problem is compounded by the fact that it is fairly common for the scene geometry to change over months of production and even parameter settings that were perfect initially might not be appropriate further down the line.

We propose to solve this problem by smoothly blending between existing fluid animations. We first develop a semi-automatic method for matching two existing liquid animations. Once we have this matching, we are able to immediately create new fluid motion that plausibly interpolates the input.

Our method allows us to instantly synthesize hundreds of new animations from a sparse set of input samples and can be used to interactively explore the parameter space. Further, since our technique is based on interpolation, it guarantees that the output will not deviate significantly from the provided inputs. After the initial precomputation phase, new animations can be generated on the fly at interactive rates. As a consequence, it has potential applications in games or training simulators where it can be used in place of a real-time fluid simulation.

6.2 Problem

We aim to interpolate between two or more existing liquid animations. A liquid animation consists of a set of closed manifold triangle meshes, with each mesh representing the liquid surface at a specified time. Each individual surface mesh may have an arbitrary number of voids, connected components, and tunnels (representing bubbles, droplets, and tumbling waves).

Naively matching two such mesh sequences together frame-by-frame is problematic, because each pair of meshes may have significantly different topologies, and because the surfaces must be temporally coherent as they evolve over time. The topology problems can in principle be solved by blending implicit surfaces [31] or more sophisticated mesh-based alignment techniques [20], but they cannot guarantee temporal coherence without additional work.

For this reason, we opt not to match together individual animation frames, but propose to match all of the frames at once. We do this by concatenating all of the triangle meshes together into a 3D hypersurface in 4D space-time as described in Chapter 5 and then performing a high-dimensional non-rigid registration (Section 6.3). We wish to deform one space-time surface to match another one while keeping the resulting deformation smooth. This strategy amounts to minimizing both a "fitting" energy and a "smoothness" energy (Section 6.4).

Once we have successfully aligned a few spacetime surfaces, we can interpolate between them to produce intermediate motion (Section 6.5). Finally, we can rapidly extract individual frames from the spacetime surface to produce the output animations (Section 5.4). Figure 22 illustrates the steps in our method, using 2D animations as input so that we can visualize the entire spacetime surface.

6.3 Registering spacetime surfaces

The registration of two fluid animations is a challenging problem for a couple of reasons. Firstly, it is highly unlikely that the spacetime mesh corresponding to the source can be mapped onto that of the target through a single rigid transformation.

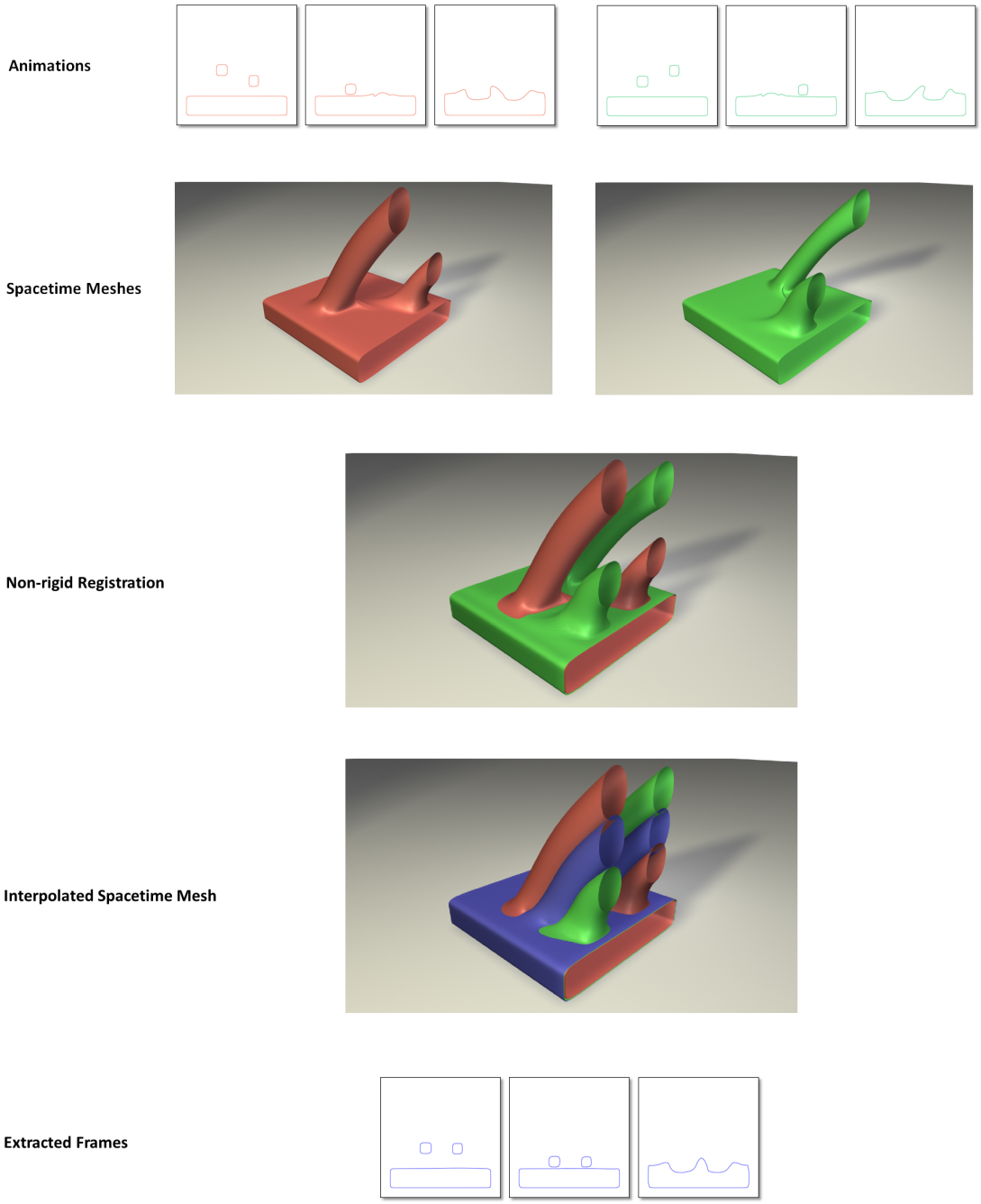

Figure 22: Overview of our method, using 2D animations for illustrative purposes. The input animations show two drops falling into a pool. The spacetime meshes are 3D surfaces that are registered to one another. A blended spacetime mesh is then created and sliced to produce final animation frames.

Secondly, due to the non-linear nature of Navier Stokes equations, the surface can stretch or compress to a large degree. As a result, even the assumption of local

rigidity is far too constraining. This also means that using an intrinsic characteristic such as Gauss curvature to automatically create correspondences between the two surfaces is not as robust as it normally would be for aligning meshes with moderate deformations.

To solve this problem, we use a non-rigid iterated closest point algorithm (non-rigid ICP). The input to this algorithm consists of two spacetime surfaces A and B. The output is a set of correspondences (one for each vertex of A) that lie on the surface B. The basic idea behind ICP is to deform the source mesh using a small number of deformation nodes such that its vertices end up at the points nearest to them (correspondences) on the target. Then, new correspondences are computed for this deformed mesh and the process is repeated until convergence.

When applied to two spacetime meshes, non-rigid ICP moves vertices of the source mesh in both space and time to best match the target. This enables us to align two animations that have seemingly contradictory constraints such as in Figure 26. This example features impact events that occur in a distinctly different order between the two input animations. Our algorithm deforms the two animations in space, and more importantly in time, to create correspondences between the two impacts. This allows us to generate an animation where both drops hit the surface simultaneously.

6.3.1 Local deformation model

In order to deform one animation into another, we use a collection of local surface deformations based on a model that was first introduced by Sumner and colleagues [124]. Here we apply it to a surface in a higher dimension. We uniformly sample the

spacetime surface with deformation nodes that are placed at a subset of the vertices. Each node $\boldsymbol{n_i}$ has an affine transform attached to it (which is split into a 4×4 matrix $\boldsymbol{A_i}$ and a 4×1 translation vector $\boldsymbol{b_i}$) and influences all vertices within a given radius (measured using geodesic distance computed by a fast marching algorithm). Hence, we have to solve for 20 unknowns per node such that the resulting surface is as close to the target as possible. The position $\boldsymbol{v_i^{k+1}}$ of a vertex at iteration $k+1$ is determined by the weighted sum of all nodes $\boldsymbol{n_j}$ that influence it:

$$\boldsymbol{v_i^{k+1}} = \sum_j w\left(\boldsymbol{n_j}, \boldsymbol{v_i^A}\right)\left(\boldsymbol{A_j}\left(\boldsymbol{v_i^k} - \boldsymbol{n_j}\right) + \boldsymbol{n_j} + \boldsymbol{b_j}\right) \tag{22}$$

where w is a weight computed based on the geodesic distance between $\boldsymbol{n_j}$ and $\boldsymbol{v_i^A}$ (i.e. the distance between the node and the vertex on the undeformed mesh).

6.3.2 Energy functions

One of the characteristics of our fluid animations is that they have a large degree of local deformation, but are already globally aligned (in that no rotation or translation is required to put them both in the same coordinate frame). We have found that an energy term enforcing rigidity, which is used in many other ICP algorithms, causes significantly worse registrations for the strongly deforming meshes we are dealing with. As a consequence, we employ only fitting and smoothness energies, which will be described in more detail below.

Fitting The fitting energy functions measure how close the currently deformed version of the source is to the target. For each point $\boldsymbol{v_i}$ on the source mesh that has a corresponding point $\boldsymbol{c_i}$ on the target, we have a point-to-point energy that computes the Euclidean distance separating them. Similarly, we also have a point-to-plane

energy that permits sliding along the plane of the corresponding point and helps to smooth out the energy landscape.

$$E_{\text{point}} = \sum_i \left\| \boldsymbol{v_i^{k+1}} - \boldsymbol{c_i} \right\|_2^2 \tag{23}$$

$$E_{\text{plane}} = \sum_i \left| \langle \boldsymbol{N_i},\ \boldsymbol{v_i^{k+1}} - \boldsymbol{c_i} \rangle \right|^2 \tag{24}$$

Smoothness The smoothness energy term ensures that affine transforms of adjacent deformation nodes are similar to each other. For each node, this energy measures the distance between its predicted position using the affine transform of its neighbor and its actual position based on its translation vector. The energy is defined as follows:

$$E_{\text{s}} = \sum_{\boldsymbol{n_i}} \sum_{\boldsymbol{n_j}} w(\boldsymbol{n_i}, \boldsymbol{n_j}) \left\| \boldsymbol{A_i} \left(\boldsymbol{n_j} - \boldsymbol{n_i} \right) + \boldsymbol{n_i} + \boldsymbol{b_i} - \left(\boldsymbol{n_j} + \boldsymbol{b_j} \right) \right\|_2^2 \tag{25}$$

6.3.3 Subsampling

Each fitting energy function for a vertex adds 4 rows to the Jacobian. Note that each deformation node has compact support, so, on average each vertex in 4D spacetime is influenced by 27 nodes. The spacetime meshes for a 5 second animation clip typically contain several million vertices, and as a result, if we used the entire mesh for registration, we would end up with a Jacobian matrix that has on the order of tens of millions of rows. Even with sparse matrices, such large sizes place unreasonable requirements on memory and add to the computational cost of the algorithm.

Instead, we subsample the mesh by randomly picking 10% of the vertices for each iteration of the algorithm. We retain 20% of the sampled vertices from the previous iteration by selecting those with the highest error. This random subsampling scheme speeds up the algorithm by close to an order of magnitude, reduces the memory footprint, and works flawlessly in practice.

6.3.4 Solving the linear system

We define the total energy of the system as a weighted sum of the energies defined above:

$$E_{\text{total}} = \gamma_{\text{s}} E_{\text{s}} + \gamma_{\text{point}} E_{\text{point}} + \gamma_{\text{plane}} E_{\text{plane}} \tag{26}$$

where the weights γ_{s}, γ_{point}, and γ_{plane} are set to 250, 0.1, and 1 in our implementation, respectively.

Since each term in E_{total} is at most quadratic in our free variables (the entries of the affine transformation matrix $\boldsymbol{A_i}$ and translation vector $\boldsymbol{b_i}$) , we can view it as a weighted least squares problem. We solve this using normal equations:

$$\left(\boldsymbol{J^T W J} + \lambda \boldsymbol{I}\right) \boldsymbol{\Delta x} = -\boldsymbol{J^T W E} \tag{27}$$

$$\boldsymbol{x^{k+1}} = \boldsymbol{x^k} + \boldsymbol{\Delta x} \tag{28}$$

where $\boldsymbol{J}$ is the Jacobian of the system, $\boldsymbol{W}$ is a diagonal weight matrix, $\boldsymbol{E}$ is the column vector created from each squared term in E_{total}, λ is a damping factor

(typically set to 10^{-6}) and $\boldsymbol{x}$ is a column vector created by concatenating all free variables from each deformation node.

The left hand side of the system is a dense, symmetric positive definite matrix. Note that regardless of the number of vertices in the mesh, the size of the resulting linear system depends only on the number of deformation nodes. For relatively small matrices, we use a Cholesky decomposition followed by backsubstition to solve the system. For large matrices ($> 10,000$ rows), we employ a preconditioned conjugate gradient solver. After solving the system, we deform the source mesh using the new affine transforms and then recompute correspondences. We repeat this for a number of iterations (usually $10-20$) or until the difference between the total energy in successive iterations falls below a threshold. Our system also allows for an incremental or iterative registration process where the user can add new correspondences to fine-tune a previous registration. To incorporate the newly added correspondence, we create a new deformation node on the source mesh at that position in spacetime. Next, we recompute the node and vertex weights for all points near newly added correspondence. We then load the previously computed values for all of the free variables and apply a few passes of diffusion on the translation vectors to ensure that the solution is smooth and resume the registration process.

6.3.5 Finding correspondences

The ICP algorithm relies on finding good correspondences at the beginning of each iteration. Given a vertex on the deformed version of the source mesh, we set its corresponding vertex to the closest point on the target whose normal points in the same direction as that at the source vertex (i.e. $\mathbf{n_{source}} \cdot \mathbf{n_{target}} > 0$). The user can

also prescribe a sparse set of correspondences as described in Section 6.4.

Since we do not have an explicit global tessellation of the spacetime surface, we first find the closest vertex on the target (using a kd-tree lookup) and then find the closest point by projecting onto the tetrahedra that surround it. To accomplish this, we construct the local surface near the vertex by creating triangular prisms for each face incident at that vertex. We can build these prisms by looking up the forward neighbor in time $(t+1)$ for each vertex of a triangular face at time t. Next, we split up each prism into three tetrahedra and find the closest point on these tetrahedra with a compatible normal. We discard points that are separated by a distance that is greater than the maximum distance between any pair of user defined correspondences.

In our experience, using the closest point rather than the closest vertex makes the registration more robust and is worth the additional computational expense.

6.3.6 Handling thin sheets and droplets

Splashing liquids often exhibit thin sheets and flying droplets (see Figure 23). These features tend to evolve very differently for even minor perturbations. In many cases, there may not be an obvious mapping between these features for two simulations. Further, if a corresponding feature does not exist in the target, it might get deformed incorrectly onto another region of the target mesh and might lead to unrealistic behaviour in the interpolated output. As a result, if the user does not explicitly provide a correspondence for a thin sheet or a droplet, we do not attempt to automatically register them onto the target.

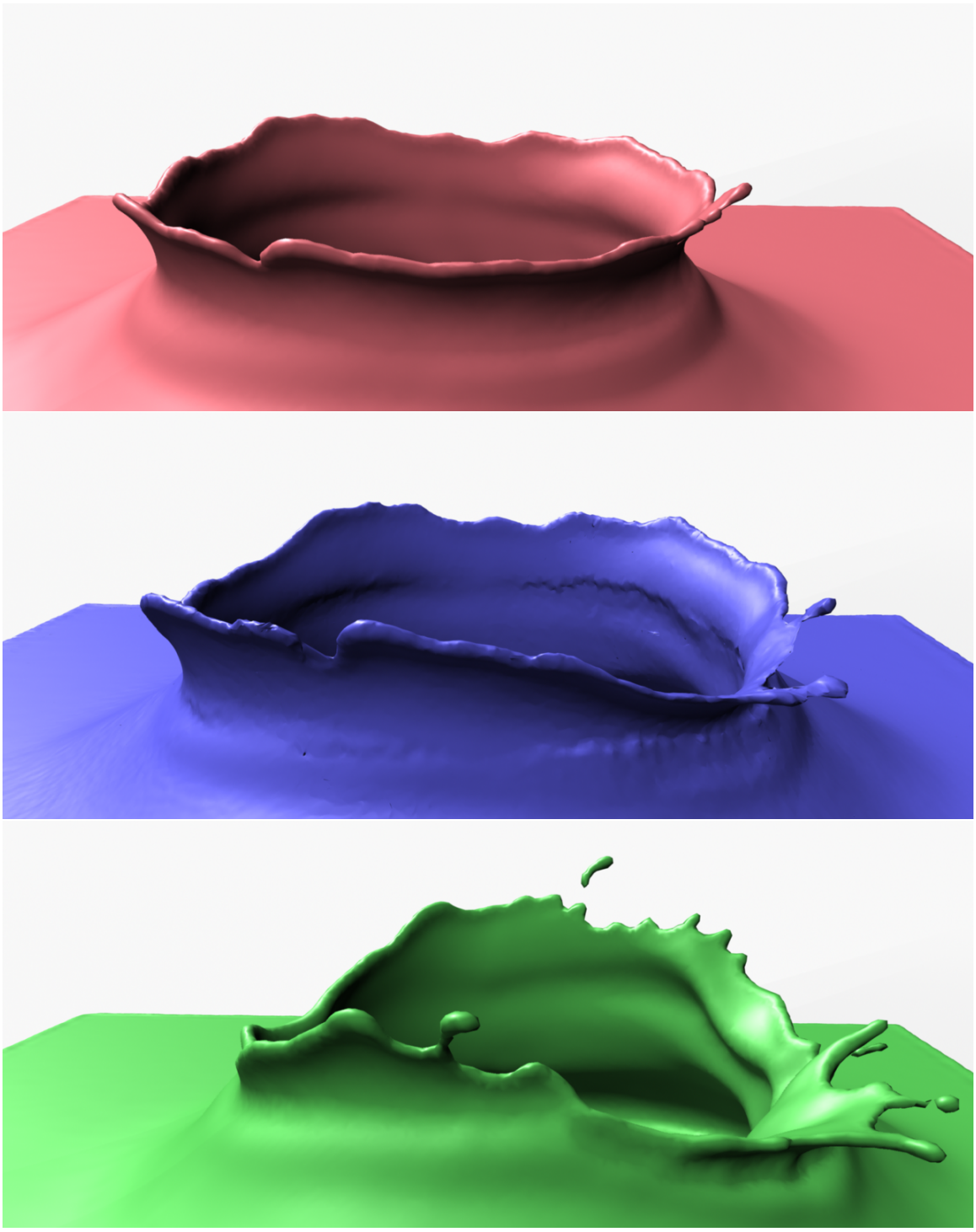

Figure 23: The 50 percent interpolation (middle) of two crown splash animations (top and bottom).

To produce plausible interpolations, we preserve such features from the source animation instead of deforming them into an incorrect portion on the target. The

easiest way to do this is to exclude from the fitting energy functions all vertices that are a part of a thin sheet or a droplet. Note that these vertices are still influenced by the nearby deformation nodes, and the smoothness energy will ensure that their behavior is consistent with the global deformation.

We perform a connected component search of the triangle mesh at each frame. Any component with fewer than 150 vertices is considered to be a droplet and all of its vertices are excluded from the solve. We preserve these drops without distorting them by finding their centroids, moving these centroids through the displacement field (computed using Equation 22) and finally reconstructing the vertices. Next, we find vertices that are part of a thin sheet. For each vertex, we query the kd-tree for the 50 nearest neighbors within a radius equal to the minimum sheet thickness (typically the width of a grid cell). If more than 10% of its nearby points have a normal that points in the opposite direction (i.e. the dot product of the two normals is negative), we flag all vertices as belonging to a thin sheet. Further, we grow out this region by two rings (i.e. all neighbouring vertices as well as their neighbours) to remove any stray vertices that might not have been identified.

6.4 User-guided registration

A drawback of the closest point search in Section 6.3.5 is that it may not always find a corresponding point that we might want. This is of particular concern because a set of poor initial correspondences can easily cause non-rigid ICP to settle on a local minimum. For instance, in Figure 26, the vertices on the top of the lower of the drops will pick their corresponding points on the top of the pool instead of those on the drop in B because of their proximity. To resolve this ambiguity, the user can provide

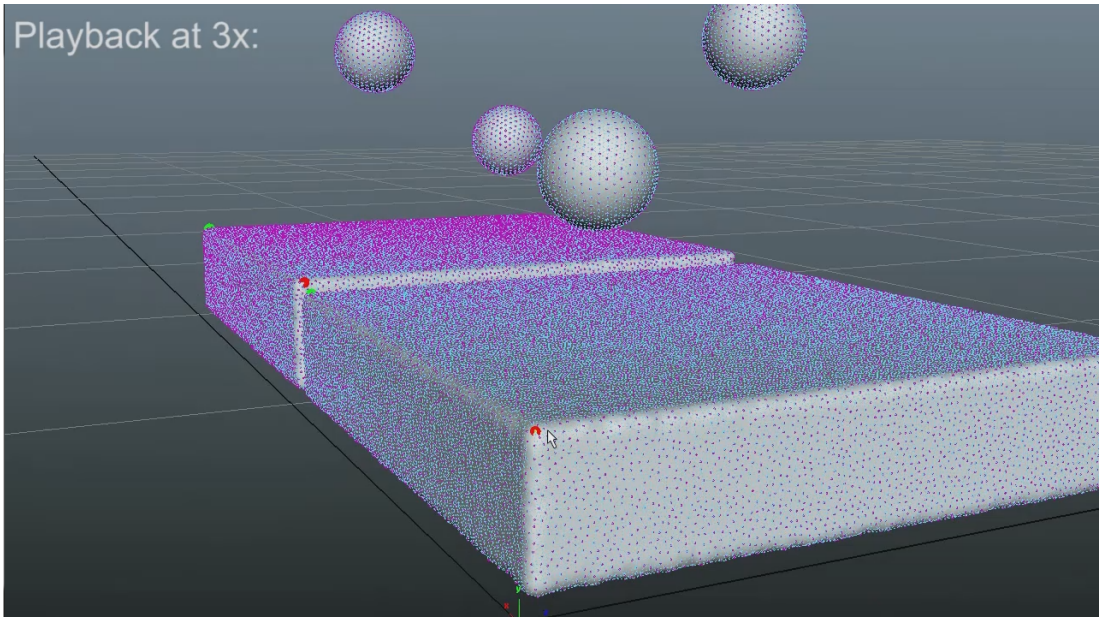

Figure 24: Our user interface for specify correspondences

a sparse set of correspondences between pairs of points on A and B (for instance, for the top and bottom points of the drop). Note that correspondences are not restricted to points that lie on the same frame (i.e. same time) in both animations and are instead specified in spacetime. As a result, we can map events that occur at different points in time to each other (such as the impacts of two droplets).

We specify user correspondences using a simple UI in Maya [11]. For a demonstration of our interface, please see Figure 24. To create a correspondence, the user first selects the type of correspondence and then selects two vertices (one on the source and the other on the target). To simplify the task, we allow three kinds of correspondences:

1. Point : The most basic form that maps one point in spacetime onto another.

2. Trajectory : This is used when one spatial feature needs to be mapped onto another for a duration of time. For instance, if one wants to specify that the

tips of the splashes need to be aligned, then a trajectory correspondence will ensure that the alignment persists until the splash dies out. If the user does not specify the length of the trajectory, the algorithm will try to preserve this correspondence for the entire animation. Note that we require only the starting points because we can use the velocity field to trace the correspondence through time.

3. Space : This pins a vertex to a point in space, but allows it to slide in time. It is useful if the two simulations have different solid boundary conditions. For instance, if we vary the width of a solid obstacle in the input simulations, then we would expect the interpolated liquid to respect the new boundary conditions and not intersect with the interpolated solid wall.

Using these three types of correspondences, the user can quickly map salient features from one simulation onto the other. Further, the user can specify whether the correspondence needs to be strictly enforced (hard) or not (soft). In general, a hard correspondence will significantly influence the output of the ICP algorithm, while a soft correspondence may be ignored if the rest of the mesh disagrees with that particular choice. On average, the user only needs to specify 10 to 20 correspondences in total for an animation consisting of 10 million spacetime vertices.

We create deformation nodes at each vertex of the source animation that is specified as a user correspondence. These correspondences affect the registration in a couple of ways. First, they are used in an initial diffusion step where the user correspondences are propagated to other deformation nodes. We fix the translation vector $\boldsymbol{b}$ for all user correspondences, and then run passes of diffusion using the vector valued heat equation on the translation vectors. This helps to provide a globally consistent

initialization for the deformed mesh. Secondly, user correspondences override the closest point search for that vertex. Soft correspondences are only applied during the diffusion step and the closest point search and do not influence other deformation nodes directly. In contrast, hard correspondences serve as equality constraints on one or more of the elements (x, y, z, t) of the translation vector $\boldsymbol{b}$ for the deformation node during the solve. This means that nearby vertices as well as deformation nodes are affected by a hard correspondence.

We implement the constraints for the hard correspondences in the following way:

1. Point: We completely specify all elements of the translation vector for the node.
2. Trajectory: We trace each point through the velocity field and create deformation nodes at uniform intervals along time (typically every 10 frames). The translation vectors of these deformation nodes are fully specified. In other words, a trajectory correspondence is a set of point correspondences for a single vertex over time.
3. Space: We create deformation nodes at the specified spatial location at uniform intervals along time and fix (x, y, z) of the translation vector while allowing it to move along time.

During the solve, all specified elements are removed as free variables and their known values are substituted into the energy functions and Jacobian.

6.5 Interpolation

Having registered the two animations, we now have a correspondence on B for each vertex of A. Given an interpolation weight α, we can produce an intermediate space-time mesh by linearly interpolating between the positions of vertices of A and their corresponding points on B. A given vertex $\boldsymbol{v_i^A}$ in mesh A will have an interpolated position $\boldsymbol{v_i^I} = (1-\alpha)\boldsymbol{v_i^A} + \alpha\boldsymbol{v_i^B}$. Note that the interpolated mesh has the same connectivity as the spacetime mesh of A because of the way the correspondences were created.

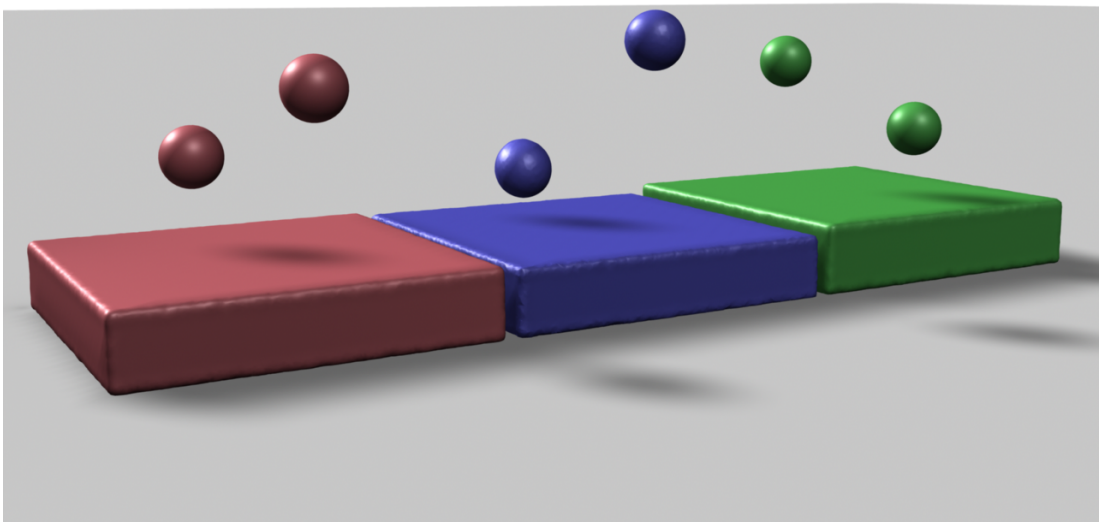

Figure 25: A frame produced using an extrapolation (middle). The interpolant value used was -0.5.

When blending between two animations, it is possible to set the blend weight α to be outside the range of zero to one. Blend weights outside this range correspond to extrapolations. We have found that for modest numerical values (e.g. $\alpha = 1.25$), such extrapolations produce plausible animation results (see Figure 25).

We can also use our blending approach to create novel animations between three

Algorithm 1 Registering liquid animations

Require: Two sequences of meshes with per-vertex velocities A, B, user correspondences u

$A_{\text{st}}, B_{\text{st}} \leftarrow$ Construct spacetime meshes from A and B.
$nodes \leftarrow$ Create deformation nodes on A_{st}.
Precompute deformation node weights for each vertex of A_{st}.
Diffuse user correspondences u to nearby deformation nodes.
while $k < maxIterations$ **do**
    $A_{defo} \leftarrow$ Deform A_{st} by using $nodes$.
    $\boldsymbol{v} \leftarrow$ Randomly subsample m vertices of A_{defo}.
    $\boldsymbol{c} \leftarrow$ Compute closest points on B_{st} for each $\boldsymbol{v}$.
    $E \leftarrow$ Compute energy vector using correspondences $\boldsymbol{c}$.
    $nodes \leftarrow$ New affine transformations from minimizing E.
    $k \leftarrow k + 1$
end while
Compute final registered mesh R from $nodes$.
return R

Algorithm 2 Interpolating liquid animations

Require: Mesh A_{st}, registration R, interpolation weight α, time t

$A_{interpol} \leftarrow (1 - \alpha)A_{\text{st}} + \alpha \cdot R$
$triangles \leftarrow$ Intersect $A_{interpol}$ with hyperplane at t.
return $triangles$

or more input animations. Assume that we have three animations, A, B, and C, and that we have a correspondence between an animation A and B, and another correspondence between A and C. Given barycentric blending weights α, β, we can produce new vertex positions according to

$$\boldsymbol{v_i^I} = (1 - \alpha - \beta)\boldsymbol{v_i^A} + \alpha\boldsymbol{v_i^B} + \beta\boldsymbol{v_i^C}. \tag{29}$$

Using this technique, our example animations can span more than just a single parameter family of animations. We can think of each example animation as a sample in a multi-dimensional parameter space. We can produce intermediate animations anywhere within the convex hull of these parameter space samples by performing barycentric blending between the three nearest surrounding samples.

During registration between spacetime meshes, we only use a loose correspondence

between adjacent frames in an animation, through time links. For surface extraction, however, we need an explicit tetrahedralization as described in Chapter 5.

After slicing the tetrahedral mesh by a hyperplane, the resulting mesh contains many sliver triangles. These sliver triangles cause noticeable shading discontinuities if they are retained. We perform one step of Laplacian smoothing [126] to even out the triangle shapes, and the resulting mesh looks considerably improved during final rendering.

Our spacetime surface registration routine is summarized in Algorithm 1, and the interpolation and surface extraction are reviewed in Algorithm 2.

6.6 Results

All of our examples were run on a workstation with an Intel Xenon E5 processor with six cores that runs at 3.2 GHz and has 72 Gbytes of main memory. The fluid simulation code and our registration code are written in C++, and they are both multithreaded. For our matrix solves, we use the Intel MKL library and Eigen. Table 3 gives simulation times (per simulation), registration times (for a one-way registration between source and target), and mesh extraction times for each of our examples. Note that the entire registration process requires less time than the time it takes to run an individual fluid simulation. Also, once the registration has been performed, new animation meshes can be produced for any blend weights in a fraction of the time it takes to perform either the simulation or registration.

We use a common color scheme in all of our examples (still images and video).

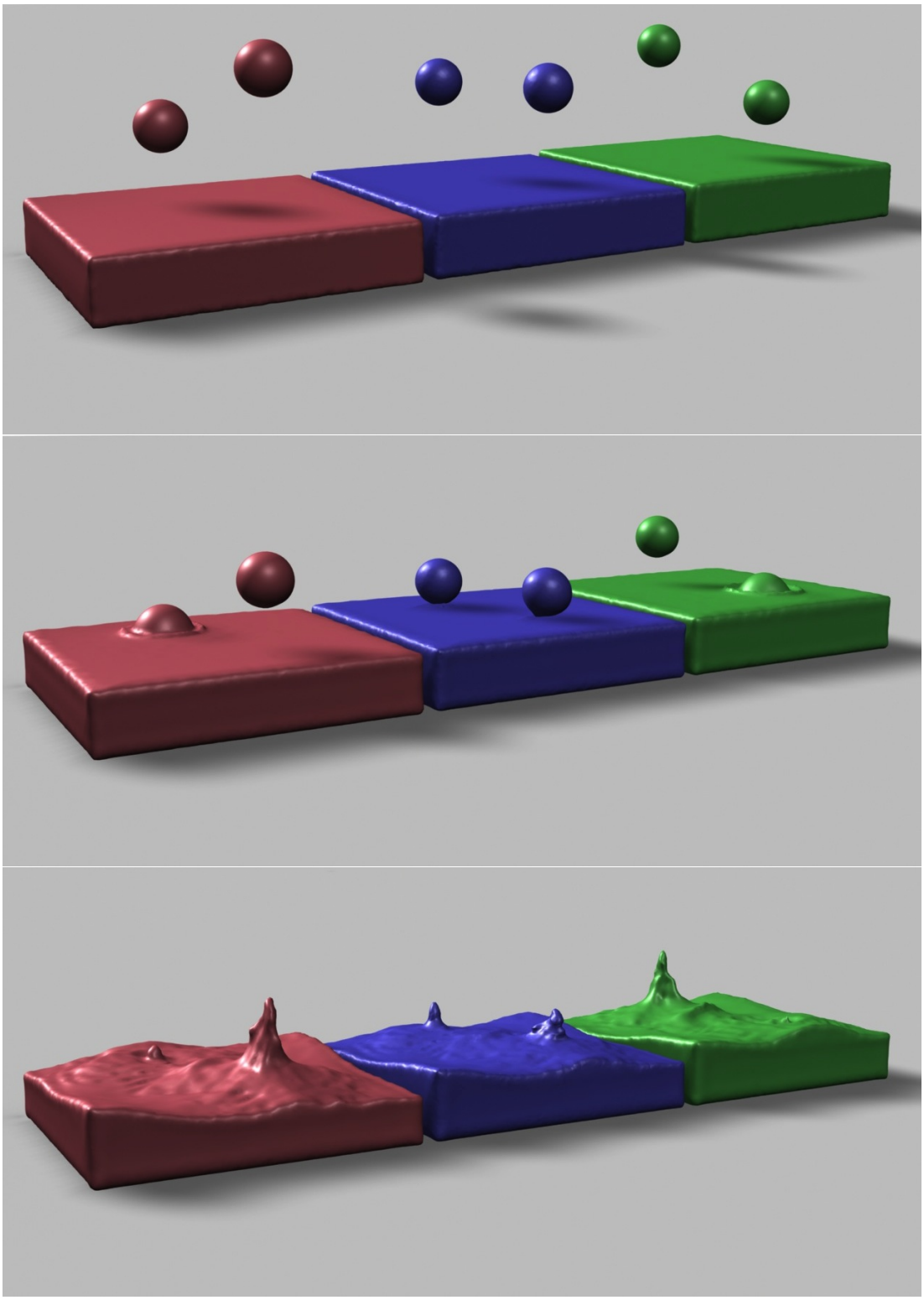

Figure 26: Animations of two spheres of water that are dropped into a pool. The red and green surfaces are from the input animations. Notice that in each, the spheres strike the surface at different times. The blue images show our blending result, which causes both spheres to strike the pool at the same time.

Table 3: This table gives the average number of correspondences per example, simulation time, registration time, and the time it takes to extract all meshes in the entire sequence. All sequences consist of 150 frames except for the crown splash (75 frames). All times are given in minutes, so we extract about 5 meshes per second for input meshes with 50k vertices.

Animation	Number of Correspond.	Simulation Time	Regist. Time	Extract Time
Two Drops	8	35.2	14.8	0.31
Dam Break	20	40.7	14.1	0.25
Duck	10	22.1	15.7	0.35
Crown Splash	20	67	22	1.9

Red and green meshes indicate original animation sequences that were generated by running a standard fluid simulation. Blue meshes indicate blended results that were created using our method. During registration, we always deform the red spacetime mesh to match that of the green.

One of the strengths of our approach is that it can alter the timing of events in an animation. An extreme example of this is demonstrated in Figure 26, where two drops of water are released and strike a pool. In one animation (red) the leftmost drop hits the water first, and in another animation (green), the rightmost drop hits first. Using our animation blending approach, we can create an entire family of animations in which the two drops hit the water at various times. For instance, the blue animation shows a variant in which the two drops strike the water simultaneously. Note that this result would not be possible if we deformed our animations purely in space and not in time.

Our dam break example shows that we can interpolate between more than two animations (Figure 28). In this set of animations, a block of water is released at one side of a long pool, and this forms a wave that sweeps the length of the pool. A wall blocks a portion of the pool, so the water must sweep around this wall. We

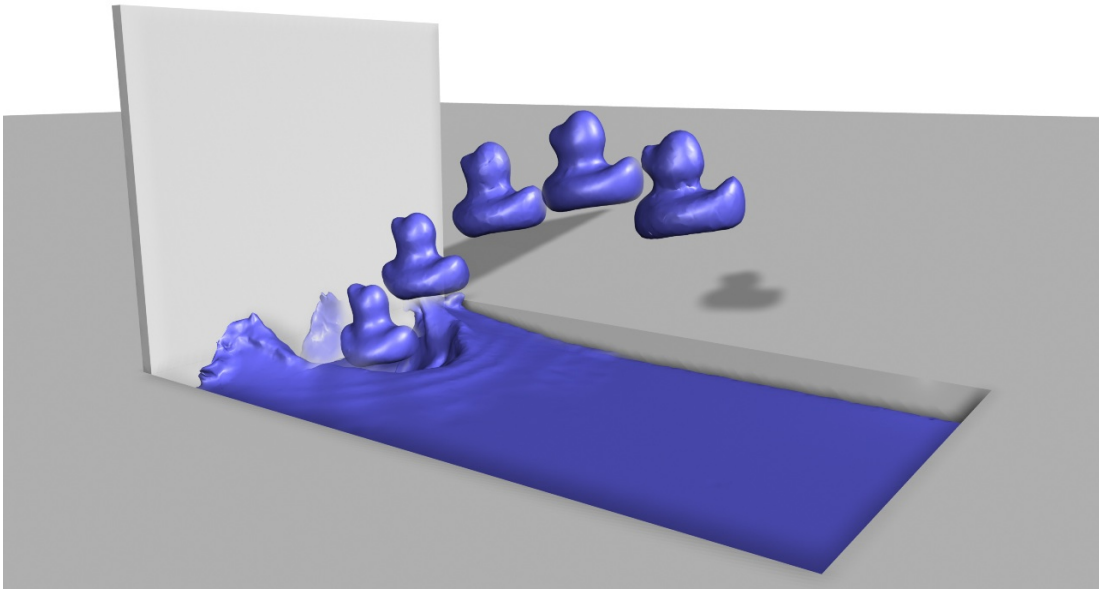

Figure 27: This composite image shows a duck being thrown into a pool of water. This animation was produced by blending between two animations of ducks that were thrown at different angles.

simulated animations for each of four different positions and widths for this wall, giving us a two-dimensional family of parameters (wall width, wall position). These two distinct parameters allow us to create new animations anywhere within this two dimensional parameter space (see Figure 29 for details of the parameter values that we used). In Figure 28, we show stills from nine different animations to show the range of variations that this allows.

The crown splash example (see Figure 23) demonstrates how our algorithm works on input that has droplets, thin sheets and numerous topology changes. Despite the relative coarse sampling of the mesh, the interpolated result captures the global behaviour of the two crowns. However, the droplets in the interpolation depend on the choice of the source animation.

We compared our interpolated results to actual simulations that were run with

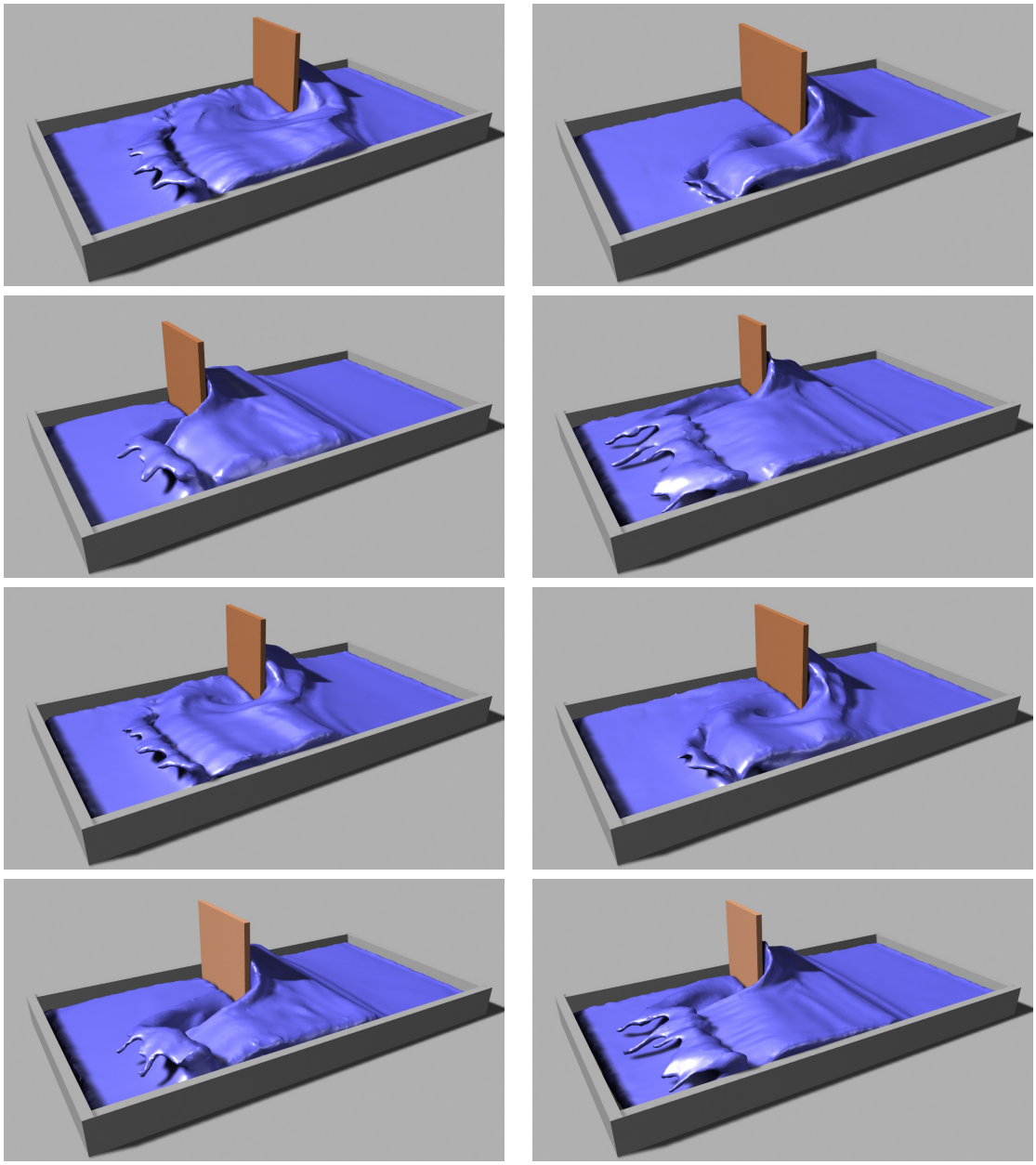

Figure 28: Snapshots from nine different animations that were created by varying the wall width and wall height of a dam break. Each image is from the same point in time. All results were created by our fluid blending method, and only four original animations were used as input.

appropriate parameter values. The two animations are qualitatively similar, however there are certain differences as can be seen in Figure 30.

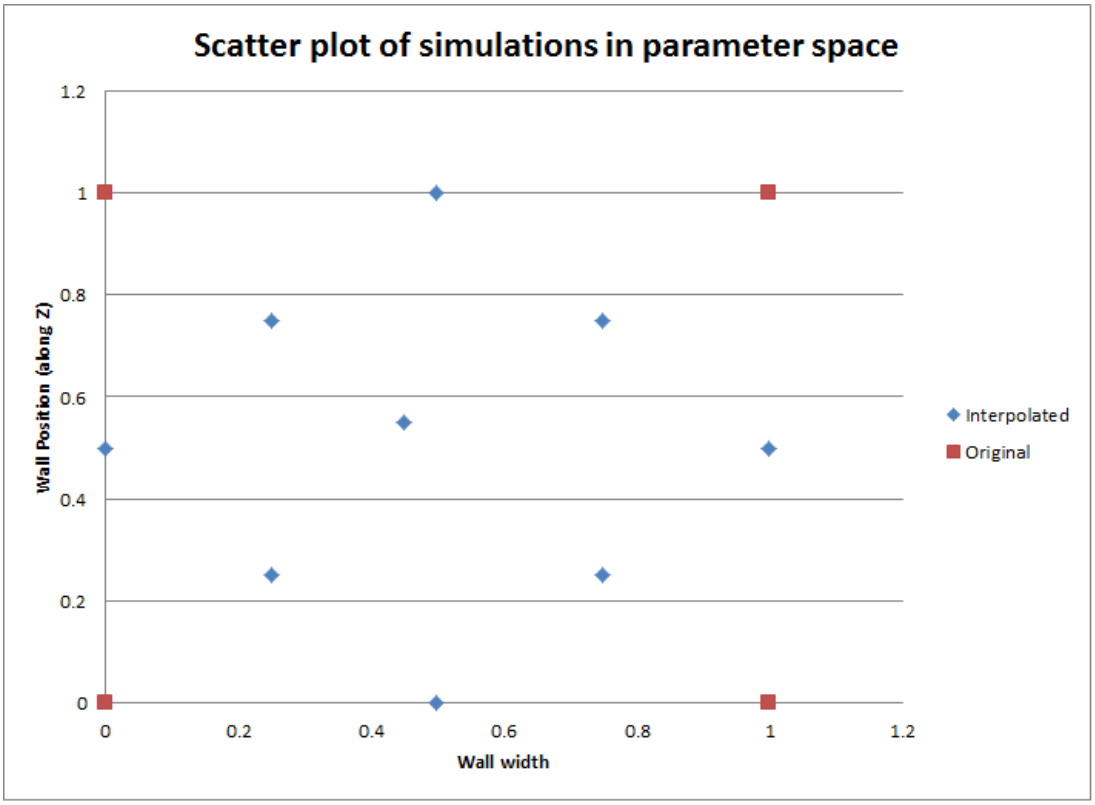

Figure 29: A scatter plot of original and interpolated simulations of the dam break in parameter space (wall width and wall position).

Figure 27 shows a fluid duck being thrown into a pool of water. The original animations have the ducks thrown at few different angles, causing them to splash into the pool at different locations. We can interpolate between any pair of input animations, allowing us to span a continuous range of possible throwing angles. In addition, we aimed two of the ducks at the wall, which adds a discontinuity to the behavior of the animation. If we select one of our input animations to capture this discontinuity, we can smoothly interpolate across this event. Interpolating between the two ducks that hit the wall allows us to produce duck strikes along a range of positions on the wall. We can even use extreme blending weights ($\alpha = 1.25$) to produce new animations in which the duck hits the wall above the highest example animation.

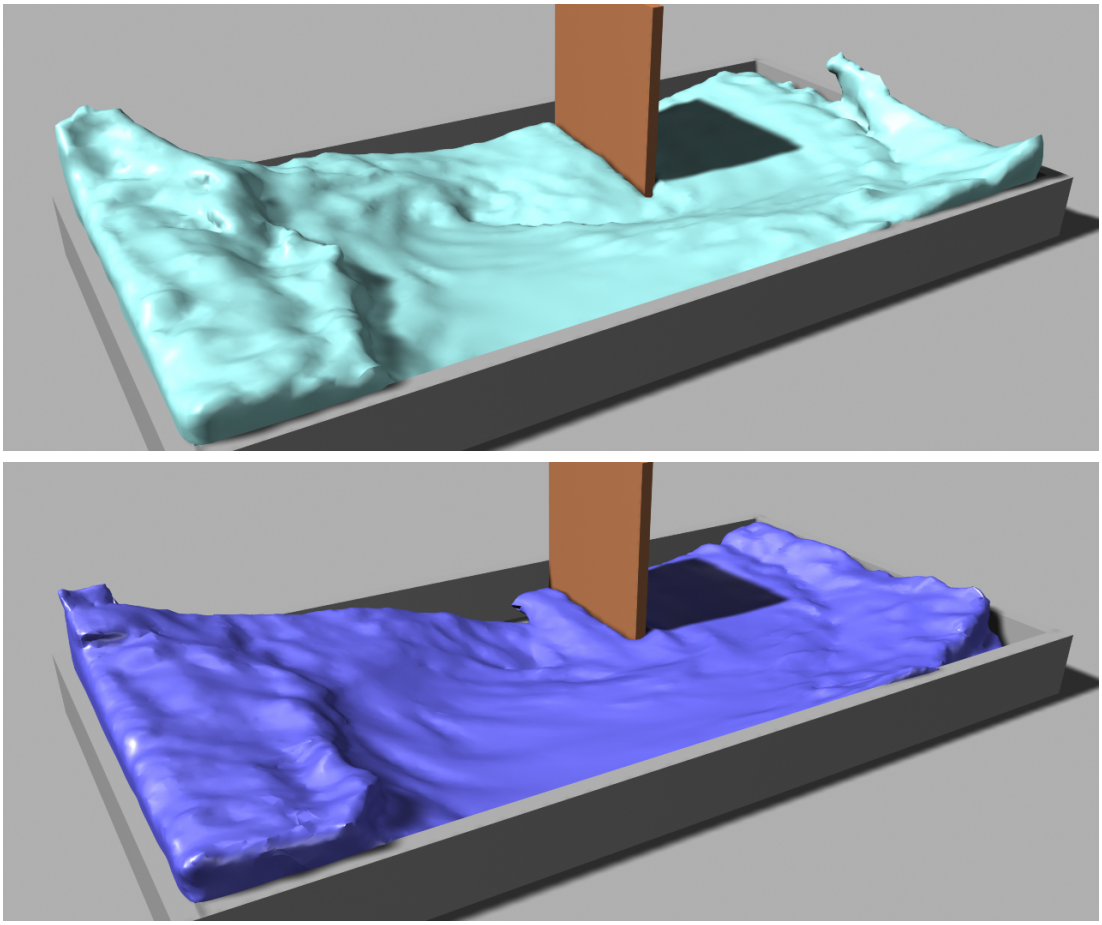

Figure 30: A comparison of the simulated result (top, in cyan) against our interpolated result (bottom, in blue) for the dambreak scenario with the wall width set to 50 percent .

6.7 Discussion and Limitations

In terms of characterizing the algorithm presented in this chapter, we look back to the criteria we defined in Chapter 1. The idea of blending between liquid animations is more useful for naturalistic animations since there is a range of behaviour between the inputs that might of interest to the animator. The workflow presented in this chapter runs contrary to most control algorithms where simulation is necessary for producing new outputs. As a result, our algorithm imposes a computational overhead (the registration step) that needs to be run exactly once between a pair of input

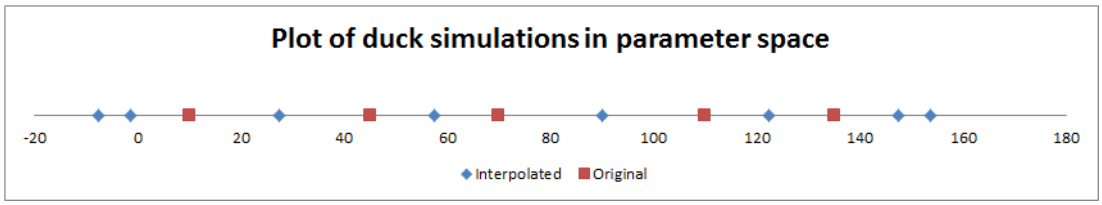

Figure 31: A scatter plot of original and interpolated simulations of the duck example in parameter space (angle made with the vertical).

animations. The registration times are comparable to those of a single simulation as long as the systems have adequate memory. Computational resources aside, we require the animator to guide this registration by annotating the inputs with correspondences between salient features and events. We believe that this process is still practical since only a handful of correspondences for each registration. Finally, in terms of constraints, with our approach, the animator is guaranteed to obtain an output that is bounded by the two specified inputs. The quality of this interpolation naturally depends on the accuracy of the registration.

Currently, our method has several limitations that suggest possible directions for future research. First, our technique is aimed at interpolating between simulations that are qualitatively similar and that do not have highly divergent behaviors. This means that we might need additional simulation samples for parameters that produce highly discontinuous behavior. For instance, our method is unlikely to produce plausible behavior near solid boundaries if none of the samples involved the solid boundary. It would be useful if we could automatically determine if two simulations are too far apart by using the energy functions.

Second, we rely on the user to identify a few salient features in each simulation. In our experience, these features coincide with regions of high curvature in space or in time. One possibility is to build up a statistical model of such features by learning

from user correspondences, and then automate the process. At the very least, we could further reduce the effort required from the users by only involving them in a simple verification step.

Another issue that we ran into was the memory consumption of the algorithm. Even with subsampling and a relatively coarse sampling of deformation nodes, registering two detailed animations requires between 30 to 50 GB of RAM. At the same time, the subsampling technique indicates that a multi-resolution approach to this problem is likely to work. It might be possible to run a coarse low resolution ICP and then break up the spacetime mesh into smaller pieces with higher sampling densities of both vertices and deformation nodes. Another implication of the coarse sampling is that small scale details of the source animation tend to be preserved in the interpolated result. The animator has a choice of designating either animation as the source. This can be seen in the

We should also note that our method can still run into the problem of local minima that is associated with non-rigid ICP. However, this has not been problematic in our tests as undesirable results can be prevented with correspondences provided by the user.

Finally, our surface extraction algorithm produces meshes that can change their triangulation fairly often. This may sometimes result in flickering especially at lower resolutions and can be ameliorated by a pass of temporal smoothing.

6.8 A guide to producing good registrations

6.8.1 Input animations

In this section, we provide some guidelines for users of our system so they can quickly produce high quality registrations. One of the most important factors that affects the registration is the set of input animations. As we have discussed earlier, our approach assumes a certain degree of similarity between the inputs. Further, the choice of the source animation has a significant impact on the output if the topologies of the input animations are quite different. In such cases, we suggest running a two-way registration and picking the appropriate animation as the source. Additionally, in our experience, a simulation that take very large steps is more difficult to deal with because the triangulation of its surface changes rapidly and topological changes tend to be abrupt. In the case of a mesh-based surface tracker, we strongly recommend picking a timestep that does not lead to the remeshing of a large portion of the mesh at each timestep.

6.8.2 Specifying correspondences

Once we have our input animations, we need to specify some correspondences between them. It is important to create correspondences between shapes that need to be preserved exactly. For instance, we would not like to see any distortion of the water drops in the falling drops example. At the same time, we must point out that it is sufficient to specify only a couple of correspondences per shape (for instance, at the north and south poles of each drop on the starting frame as well as at the point of impact with the pool). In general, hard correspondences are preferable to

soft ones unless there is a large degree of uncertainty about the quality of the correspondence. When it comes to choosing the type of correspondences, we think that point correspondences serve as a good default option.

Line correspondences may be preferred when there are features that are persistent in time. Some good examples of this are the corners of the pools and the tips of splashes. Since line correspondences are converted to point correspondences within the program, they merely simplify the work for the user during the specification step.

Space correspondences are handy when the input simulations have different static solid boundary conditions. However, unlike line correspondences which can move in space due to the velocity field, these space correspondences stay fixed and ensure that the solid boundaries are respected in the interpolation. We used this kind of correspondence in our dam break example where the width and position of the wall was varied. We must point out that if the solid happens to be in motion, we would need to modify our implementation to account for its velocity and map to the appropriate point in the target animation.

6.8.3 Sampling

Finally, the user needs to select a sampling rate for the deformation nodes as well as subsampling rate for the vertices. The former dictates the degrees of freedom available to deform the mesh while the latter affects the memory footprint and accuracy because it determines which vertices are used to evaluate the fitting energy function. In our experiments, we picked a sampling rate such that each deformation node affected 200 to 400 vertices. We found that we could achieve acceptable results by setting the

subsampling rate as low as 1/30. If memory is not a constraint, then we recommend a subsampling rate of 1 (i.e. the energy is computed for every vertex).

6.8.4 Failure cases

When we run a registration and evaluate the resulting interpolated animation, we might find that it either contains artifacts or that it does not look *right*. This could be due to a number of factors. One of the possibilities is that a correspondence was specified incorrectly, resulting in a large deformation of the surface. Our Maya plugin allows the user to select vertices on the source animation and view their corresponding points on the target animation. If there is an obvious mismatch, then the user can fix the incorrect correspondences by either modifying them or deleting them altogether.

Similarly, it is possible that the user might specify a correspondence that causes an unnatural speedup or slowdown of a portion of the animation. Again, in this case, the solution is to find the problematic correspondence, fix it and re-run the registration.

Other common failure cases include situations where we find that a specified correspondence had a more global effect than was intended. For instance, while mapping two splashes that occur at different points of the domain, if we forget to pin down the boundary of our domain, we could see a global shifting of the water body. This can be easily remedied by fixing regions of the mesh using additional point or line correspondences.

A failure mode that can be trickier to resolve is one where the interpolation

resembles the source more closely than it does the target regardless of the choice of the interpolant. While increasing the number of deformation nodes can allow the surface to deform to a greater extent, this does not always guarantee that the resulting interpolation remains plausible. This is of particular concern when the two input animations are sufficiently different. It is possible to ameliorate this by carefully specifying additional point correspondences or by running a new simulation that is closer to the source. Detecting such cases using the energy functions is a promising avenue for future work.

6.9 Summary

In this chapter, we introduced a flexible new method for the generation and control of liquid animations. We can instantly generate new fluid animations on the fly by simply interpolating between existing simulations, and we can plausibly fine-tune a simulation by warping its spacetime representation. In the future, we envision a fully automated version of our system that generates an infinite set of simulations from a given range of input parameters. We believe a system like this would make a significant impact on the state-of-the-art in special effects production, and it could easily generalize to other phenomena beyond liquids.

CHAPTER VII

CONCLUSION

In this final chapter, we summarize the ideas introduced in this thesis and present some preliminary results in areas that we think are fruitful avenues for future investigation.

7.1 Discussion

By using a mesh as the fundamental representation of our liquids (during input specification, simulation and manipulation), we have been able to control liquids in a couple of ways:

In Chapter 4, we developed a method for producing liquid animations when the animator has a clear idea of what shapes they must form. We looked at ways to simplify the input that the animator needs to provide by introducing a simple way to preserve volume while morphing between a set of keyframes. In this application, by representing the in-betweens as meshes that retained the connectivity and topology of the inputs, we were able to invoke the divergence theorem and perform the quadrature over the discrete set of triangles. We then came up with an efficient control scheme that used a mesh-based representation for both inputs and the liquid surface. This choice of representation opened up a number of possibilities for the animator to dictate what the output of the simulator ought to be. Compared to other options such as

level sets or particles, the mesh-based surface maintains a great degree of coherence between frames and this enables us to perform Lagrangian simulations of waves and springs on it.

In Chapter 6, we eschewed the idea of using keyframes for control, and instead introduced a new workflow for dealing with liquid simulations by enabling the animator to blend existing animations of liquids. This solves a number of issues that other methods have not been able to address. First, it removes the need for re-simulation and replaces it with a precomputation step. Secondly, it allows for near-realtime visualization of any animation that spans the set of inputs and lastly, it guarantees that this output will lie between the provided inputs. The building block for this algorithm is a spacetime mesh, which is a Lagrangian representation of an animation embedded in a higher dimension. By performing a non-rigid registration of a spacetime mesh onto another, we obtain per-vertex correspondences between the two and this allows us to interpolate between them. Here again, by using a mesh to represent the liquid surface, we are able to construct the corresponding spacetime surface in a very natural manner by simply tracing vertices through time. Further, we are able to easily pick samples on the surface (a subset of the mesh vertices) and compute normals at these locations because these are well defined on a mesh. Finally, the largely-consistent connectivity of the surface enables efficient surface extraction from the spacetime mesh.

7.2 Future work

Fluid simulation and control have taken significant strides forward in the last decade. While CPU clock speeds have not increased dramatically, the advent of massive parallelism both in the consumer and enterprise sectors opens up a range of possibilities for future research. As GPUs start to support highly parallel general purpose computation, we are likely to see interactive and possibly real-time editing and control of simulations. On the server side, we should expect to routinely run numerous simulations simultaneously on a number of CPUs. We have already seen advances in terms of precomputed dynamics produced by hours of computation on a cluster of machines [64].

7.2.1 Interpolation in a larger parameter space

In this thesis, we explored interpolation between animations that were produced by varying two parameters at most - for instance, the dam break example in Chapter 6 allowed the user to pick the position of the wall along the Z axis and also vary its width at the same time. In order to construct this example, we ran four simulations and varied the two parameters to obtain a reasonable sampling of the parameter space. Can we scale this up to a higher number of parameters? Undoubtedly, the curse of dimensionality is likely to strike at such parameter counts since we require pairwise registrations between the simulations that we intend to interpolate between. However, it is reasonable to expect an animator to change a small number of parameters (< 6).

Ideally, we would want to avoid burdening the animator with the task of sampling

the parameter space. We would like to sample as sparsely as possible while producing plausible interpolated results. This means that our sampling algorithm has to be aware of the limitations of the registration technique. Any parameter that involves the boundary conditions is likely to need careful attention because of potential discontinuities. In the case of the duck example, we manually picked samples near the point of impact with the wall in order to capture the correct behaviour. In order to automate the process, we need to come up with certain metrics that capture the discrepancy between two simulations. It might be possible to detect events such as impact of water against a solid boundary or sharp changes in velocity. We might also be able to draw upon techniques used in computer vision such as shape descriptors to come up with analogues for spacetime surfaces. Another option is to look at the fitting energies after several iterations of the non-rigid registration algorithm to determine if the simulations are too far apart.

7.2.2 Semi-automatic correspondence generation

A related issue is that of specifying user correspondences. Currently, the user manually specifies a small number of correspondences to guide the registration by selecting pairs of points. As the number of simulations grows larger, this correspondence specification step is likely to become a bottleneck in the pipeline. It would be extremely useful if the system could suggest a set of correspondences to the user, who can then either verify or modify them instead of specifying them from scratch. Intrinsic characteristics of the surface such as curvature in space and time might be a good starting point for detecting points of interest. We think that it might also be possible to learn what an interesting feature is by applying machine leaning techniques to a set of labeled simulations. Further, it would be extremely valuable if a set of correspondences

from one pair of animations could be transferred over to another. This would be particularly useful if the animator decides that a simulation needs to be re-run or if only small changes are required to map slightly different simulations resulting from changes to parameters.

7.2.3 Library of animations

Ultimately, the goal is to use our framework to create a continuous library of liquid animations through an automatic sampling of the parameter space. The animator would specify the geometry in the scene and initial boundary conditions for the liquid as well as parameters that he might like to modify such as positions of objects, forces or initial velocities. By running simulations in parallel, followed by pairwise registrations, we would be able to interpolate between the sampled simulations at interactive rates. Further, such a system could also take in user input to (re)-sample a certain region more densely if registrations are not of sufficiently high quality.

7.2.4 Spacetime editing of animations

In this dissertation, we deformed spacetime meshes to create correspondences between animations. This deformation was guided by user-provided correspondences. However, we think that a number of techniques from geometric processing such as spline based deformation and Laplacian surface editing are likely to generalize to these spacetime meshes. This should allow us to perform interesting operations such as retiming and editing of the output of simulations.

REFERENCES

[1] Adams, B., Pauly, M., Keiser, R., and Guibas, L. J., "Adaptively sampled particle fluids," in *ACM SIGGRAPH 2007 papers*, p. 48, 2007.

[2] Akinci, N., Ihmsen, M., Akinci, G., Solenthaler, B., and Teschner, M., "Versatile Rigid-fluid Coupling for Incompressible SPH," *ACM Trans. Graph.*, vol. 31, pp. 62:1–62:8, July 2012.

[3] Alexa, M., "Recent advances in mesh morphing," *Computer Graphics Forum*, vol. 21, no. 2, pp. 173–196, 2002.

[4] Amberg, B., Romdhani, S., and Vetter, T., "Optimal step nonrigid ICP algorithms for surface registration," in *IEEE Conf. Computer Vision and Pattern Recognition, CVPR '07*, pp. 1–8, 2007.

[5] Ando, R., Thuerey, N., and Tsuruno, R., "Preserving Fluid Sheets with Adaptively Sampled Anisotropic Particles," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18 (8), pp. 1202–1214, 2012.

[6] Ando, R., Thuerey, N., and Wojtan, C., "Highly Adaptive Liquid Simulations on Tetrahedral Meshes," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 32 (4), p. 10, August 2013.

[7] Angst, R., Thuerey, N., Botsch, M., and Gross, M., "Robust and efficient wave simulations on deforming meshes," *Computer Graphics Forum*, vol. 27, no. 7, pp. 1895–1900, 2008.

[8] Anguelov, D., Srinivasan, P., cheung Pang, H., and Koller, D., "The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces," in *In TR-SAIL-2004-100, at http://robotics.stanford.edu/?drago/cc/tr100.pdf*, pp. 33–40, 2004.

[9] Aubert, F. and Bechmann, D., "Animation by deformation of space-time objects," in *Computer Graphics Forum*, vol. 16, pp. C57–C66, Wiley Online Library, 1997.

[10] Aubert, F. and Bechmann, D., "Volume-preserving space deformation," *Computers and Graphics*, vol. 21, no. 5, pp. 625 – 639, 1997.

[11] Autodesk, "Maya software," 2013.

[12] Batty, C., Bertails, F., and Bridson, R., "A fast variational framework for accurate solid-fluid coupling," *ACM Trans. Graph.*, vol. 26, July 2007.

[13] Batty, C. and Bridson, R., "Accurate viscous free surfaces for buckling, coiling, and rotating liquids," in *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation*, pp. 219–228, July 2008.

[14] Batty, C. and Houston, B., "A simple finite volume method for adaptive viscous liquids," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pp. 111–118, ACM, 2011.

[15] Batty, C., Xenos, S., and Houston, B., "Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids," in *Proceedings of Eurographics*, 2010.

[16] Becker, M. and Teschner, M., "Weakly compressible sph for free surface flows," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, (Aire-la-Ville, Switzerland, Switzerland), pp. 209–217, Eurographics Association, 2007.

[17] Besl, P. J. and McKay, N. D., "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, pp. 239–256, Feb. 1992.

[18] Bhattacharya, H., Gao, Y., and Bargteil, A. W., "A level-set method for skinning animated particle data," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aug 2011.

[19] Blanz, V. and Vetter, T., "A morphable model for the synthesis of 3d faces," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, (New York, NY, USA), pp. 187–194, ACM Press/Addison-Wesley Publishing Co., 1999.

[20] Bojsen-Hansen, M., Li, H., and Wojtan, C., "Tracking surfaces with evolving topology," *ACM Trans. Graph.*, vol. 31, pp. 53:1–53:10, July 2012.

[21] Boyd, L. and Bridson, R., "Multiflip for energetic two-phase fluid simulation," *ACM Trans. Graph.*, vol. 31, no. 2, pp. 16:1–16:12, 2012.

[22] Breen, D. E. and Whitaker, R. T., "A level-set approach for the metamorphosis of solid models," *IEEE Trans. Visualization and Computer Graphics*, vol. 7, pp. 173–192, Apr. 2001.

[23] Bridson, R., *Fluid Simulation for Computer Graphics*. AK Peters/CRC Press, 2008.

[24] Bridson, R., Houriham, J., and Nordenstam, M., "Curl-noise for procedural fluid flow," in *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[25] Brochu, T., Batty, C., and Bridson, R., "Matching fluid simulation elements to surface geometry and topology," *ACM Trans. Graph.*, vol. 29, pp. 47:1–47:9, July 2010.

[26] Brown, B. and Rusinkiewicz, S., "Global non-rigid alignment of 3-D scans," *ACM Trans. Graph.*, vol. 26, Aug. 2007.

[27] Bruins, G. and Reisch, J., "Rat-sized water effects in ratatouille," in *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[28] Carlson, D., "Wave displacement effects for surf's up," in *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[29] Chentanez, N., Feldman, B. E., Labelle, F., O'Brien, J. F., and Shewchuk, J. R., "Liquid simulation on lattice-based tetrahedral meshes," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 219–228, Eurographics Association, 2007.

[30] Clausen, P., Wicke, M., Shewchuk, J., and O'Brien, J., "Simulating liquids and solid-liquid interactions with Lagrangian meshes," *ACM Trans. Graph.*, 2013.

[31] Cohen-Or, D., Solomovic, A., and Levin, D., "Three-dimensional distance field metamorphosis," *ACM Trans. Graph.*, vol. 17, pp. 116–141, Apr. 1998.

[32] Engel, K., Hadwiger, M., Kniss, J., Lefohn, A., Rezk-Salama, C., and Weiskopf, D., "Real-time volume graphics," 2004.

[33] Enright, D., Nguyen, D., Gibou, F., and Fedkiw, R., "Using the Particle Level Set Method and a Second Order Accurate Pressure Boundary Condition for Free-Surface Flows," *Proc. Joint Fluids Engineering Conference*, 2003.

[34] Enright, D., Losasso, F., and Fedkiw, R., "A fast and accurate semi-Lagrangian particle level set method," *Comput. Struct.*, vol. 83, no. 6-7, pp. 479–490, 2005.

[35] Erleben, K., Misztal, M. K., and Bærentzen, J. A., "Mathematical foundation of the optimization-based fluid animation method," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, (New York, NY, USA), pp. 101–110, ACM, 2011.

[36] Fattal, R. and Lischinski, D., "Target-driven smoke animation," *ACM Trans. Graph.*, vol. 23, pp. 441–448, August 2004.

[37] Fedkiw, R., Stam, J., and Jensen, H. W., "Visual simulation of smoke," in *Proceedings of SIGGRAPH 2001* (Fiume, E., ed.), Computer Graphics Proceedings, Annual Conference Series, pp. 15–22, ACM, ACM Press / ACM SIGGRAPH, 2001.

[38] Feldman, B. E., O'Brien, J. F., and Klingner, B. M., "Animating gases with hybrid meshes," *ACM Trans. Graph.*, vol. 24, pp. 904–909, July 2005.

[39] Foster, N. and Metaxas, D., "Realistic animation of liquids," *Graphical models and image processing*, vol. 58, no. 5, pp. 471–483, 1996.

[40] Foster, N. and Fedkiw, R., "Practical animation of liquids," in *Proc. of ACM SIGGRPAH*, pp. 23–30, 2001.

[41] Foster, N. and Metaxas, D., "Controlling fluid animation," in *Proceedings of the 1997 Conference on Computer Graphics International*, CGI '97, pp. 178–188, 1997.

[42] Foster, N. and Metaxas, D., "Controlling fluid animation," in *Proc. of CGI*, 1997.

[43] Gelfand, N., Ikemoto, L., Rusinkiewicz, S., and Levoy, M., "Geometrically stable sampling for the ICP algorithm," in *Int. Conference on 3D Digital Imaging and Modeling (3DIM)*, Oct. 2003.

[44] Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H., "Robust global registration," in *Symposium on Geometry Processing*, SGP '05, Eurographics Association, 2005.

[45] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H., "A schur-complement method for sparse quadratic programming," Tech. Rep. SOL 87-12, Stanford University, Systems Optimization Laboratory, December 1987.

[46] Golub, G. H. and Van Loan, C. F., *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*. The Johns Hopkins University Press, 3rd ed., Oct. 1996.

[47] Gould, N. I. M., Hribar, M. E., and Nocedal, J., "On the solution of equality constrained quadratic programming problems arising in optimization," *SIAM J. Sci. Comput.*, vol. 23, pp. 1376–1395, Apr. 2001.

[48] Guennebaud, G. and Gross, M., "Algebraic point set surfaces," in *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[49] Gurung, T., Luffel, M., Lindstrom, P., and Rossignac, J., "Lr: Compact connectivity representation for triangle meshes," *ACM Trans. Graph.*, vol. 30, pp. 67:1–67:8, July 2011.

[50] Hähnel, D., Thrun, S., and Burgard, W., "An extension of the ICP algorithm for modeling nonrigid objects with mobile robots," in *Proc. of the Int. Joint Conference on Artificial Intelligence*, (Acapulco, Mexico), IJCAI, 2003.

[51] Harlow, F. and Welch, E., "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *Phys. Fluids*, vol. 8 (12), pp. 2182–2189, 1965.

[52] Hirota, G., Maheshwari, R., and Lin, M. C., "Fast volume-preserving free form deformation using multi-level optimization," in *Proc. Symposium on Solid Modeling and Applications*, pp. 234–245, ACM Press, 1999.

[53] Hong, J.-m. and Kim, C.-h., "Controlling fluid animation with geometric potential," *Computer Animation and Virtual Worlds*, vol. 15, no. 3-4, pp. 147–157, 2004.

[54] Hong, W., House, D. H., and Keyser, J., "Adaptive particles for incompressible fluid simulation," *Vis. Comput.*, vol. 24, pp. 535–543, July 2008.

[55] Hong, W., House, D. H., and Keyser, J., "An adaptive sampling approach to incompressible particle-based fluid.," in *TPCG* (Tang, W. and Collomosse, J. P., eds.), pp. 69–76, Eurographics Association, 2009.

[56] Huang, R., Melek, Z., and Keyser, J., "Preview-based sampling for controlling gaseous simulations," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 177–186, ACM, 2011.

[57] Hughes, T. J., Franca, L., and Balestra, M., "A new finite element formulation for computational fluid dynamics: V. circumventing the babuka-brezzi condition: a stable petrov-galerkin formulation of the stokes problem accommodating equal-order interpolations," *Computer Methods in Applied Mechanics and Engineering*, vol. 59(1), pp. 85–99, 1986.

[58] Irving, G., Schroeder, C., and Fedkiw, R., "Volume conserving finite element simulations of deformable models," *ACM Trans. Graph.*, vol. 26, July 2007.

[59] Jobson, D. J., Rahman, Z., and Woodell, G. A., "Retinex image processing: Improved fidelity to direct visual observation," in *Proceedings of the IS&T Fourth Color Imaging Conference: Color Science, Systems, and Applications*, vol. 4, pp. 124–125, The Society for Imaging Science and Technology, 1995.

[60] Ju, T., Schaefer, S., Warren, J., and Desbrun, M., "A geometric construction of coordinates for convex polyhedra using polar duals," in *Proceedings of the third Eurographics symposium on Geometry processing*, SGP '05, (Aire-la-Ville, Switzerland, Switzerland), Eurographics Association, 2005.

[61] Kartch, D., *Efficient Rendering and Compression for Full-Parallax Computer-Generated Holographic Stereograms*. PhD thesis, Cornell University, 2000.

[62] Kim, B., Liu, Y., Llamas, I., Jiao, X., and Rossignac, J., "Simulation of bubbles in foam with the volume control method," in *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[63] Kim, B., Liu, Y., Llamas, I., and Rossignac, J., "Flowfixer: Using bfecc for fluid simulation," *EG Workshop on Natural Phenomena*, pp. 51–56, 2005.

[64] Kim, D., Koh, W., Narain, R., Fatahalian, K., Treuille, A., and O'Brien, J. F., "Near-exhaustive precomputation of secondary cloth effects," *ACM Transactions on Graphics*, vol. 32, pp. 87:1–7, July 2013. Proceedings of ACM SIGGRAPH 2013, Anaheim.

[65] Kim, T. and Delaney, J., "Subspace fluid re-simulation," *ACM Trans. Graph.*, vol. 32, pp. 62:1–62:9, July 2013.

[66] Kim, T., Thuerey, N., James, D., and Gross, M., "Wavelet Turbulence for Fluid Simulation," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 27 (3), p. 6, August 2008.

[67] Klein, A. W., Sloan, P.-P. J., Finkelstein, A., and Cohen, M. F., "Stylized video cubes," in *ACM SIGGRAPH Symposium on Computer Animation*, pp. 15–22, July 2002.

[68] Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F., "Fluid animation with dynamic meshes," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 820–825, 2006.

[69] Kraevoy, V. and Sheffer, A., "Cross-parameterization and compatible remeshing of 3d models," in *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 861–869, ACM, 2004.

[70] Kwatra, V. and Rossignac, J., "Space-time surface simplification and edge-breaker compression for 2d cel animations," *Int. Journal of Shape Modeling*, vol. 8, no. 2, pp. 119–137, 2002.

[71] Labelle, F. and Shewchuk, J. R., "Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles," *ACM Trans. Graph.*, vol. 26, July 2007.

[72] Landis, H., "Global illumination in production." ACM SIGGRAPH 2002 Course #16 Notes, July 2002.

[73] Lee, A., Dobkin, D., Sweldens, W., and Schröder, P., "Multiresolution mesh morphing," in *Proceedings of SIGGRAPH*, vol. 99, pp. 343–350, 1999.

[74] Lee, J. and Kry, P., eds., *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, Eurographics Association, 2012.

[75] Lenaerts, T., Adams, B., and Dutré, P., "Porous flow in particle-based fluid simulations," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pp. 1–8, ACM, 2008.

[76] Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D., "The digital michelangelo project," in *Proceedings of SIGGRAPH 2000* (Akeley, K., ed.), Computer Graphics Proceedings, Annual Conference Series, (New York), pp. 131–144, ACM, ACM Press / ACM SIGGRAPH, 2000.

[77] Lew, A. J. and Buscaglia, G. C., "A discontinuous-galerkin-based immersed boundary method," *International Journal for Numerical Methods in Engineering*, vol. 76, no. 4, pp. 427–454, 2008.

[78] Li, H., Adams, B., Guibas, L., and Pauly, M., "Robust single-view geometry and motion reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5, p. 175, 2009.

[79] Li, H., Adams, B., Guibas, L. J., and Pauly, M., "Robust single-view geometry and motion reconstruction," *ACM Trans. Graph.*, vol. 28, pp. 175:1–175:10, Dec. 2009.

[80] Li, H., Luo, L., Vlasic, D., Peers, P., Popović, J., Pauly, M., and Rusinkiewicz, S., "Temporally coherent completion of dynamic shapes," *ACM Trans. Graph.*, vol. 31, pp. 2:1–2:11, Feb. 2012.

[81] Losasso, F., Fedkiw, R., and Osher, S., "Spatially adaptive techniques for level set methods and incompressible flow," *Computers and Fluids*, vol. 35, p. 2006, 2005.

[82] Losasso, F., Gibou, F., and Fedkiw, R., "Simulating water and smoke with an octree data structure," *ACM Trans. Graph.*, vol. 23, pp. 457–462, Aug. 2004.

[83] McNamara, A., Treuille, A., Popović, Z., and Stam, J., "Fluid control using the adjoint method," *ACM Trans. Graph.*, vol. 23, pp. 449–456, August 2004.

[84] Misztal, M. K., Erleben, K., Bargteil, A., Fursund, J., Christensen, B. B., Bærentzen, J. A., and Bridson, R., "Multiphase flow of immiscible fluids on unstructured moving meshes," in *Proc. Symposium on Computer Animation*, SCA '12, pp. 97–106, Eurographics Association, 2012.

[85] Misztal, M. K., Bridson, R., Erleben, K., Bærentzen, J. A., and Anton, F., "Optimization-based fluid simulation on unstructured meshes.," in *VRIPHYS*, pp. 11–20, Eurographics Association, 2010.

[86] Mullen, P., Crane, K., Pavlov, D., Tong, Y., and Desbrun, M., "Energy-preserving integrators for fluid animation," *ACM Trans. Graph.*, vol. 28, pp. 38:1–38:8, July 2009.

[87] Müller, M., Charypar, D., and Gross, M., "Particle-based fluid simulation for interactive applications," in *Proc. Symposium on Computer Animation*, pp. 154–159, Eurographics Association, 2003.

[88] Müller, M., Keiser, R., Nealen, A., Pauly, M., Gross, M., and Alexa, M., "Point based animation of elastic, plastic and melting objects," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, pp. 141–151, Eurographics Association, 2004.

[89] Narain, R., Sewall, J., Carlson, M., and Lin, M. C., "Fast animation of turbulence using energy transport and procedural synthesis," *ACM SIGGRAPH Asia papers*, p. Article 166, 2008.

[90] Nielsen, M. B. and Bridson, R., "Guide shapes for high resolution naturalistic liquid simulation," in *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pp. 83:1–83:8, 2011.

[91] Nielsen, M. B., Christensen, B. B., Zafar, N. B., Roble, D., and Museth, K., "Guiding of smoke animations through variational coupling of simulations at different resolutions," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 217–226, 2009.

[92] O'Brien, J. F. and Hodgins, J. K., "Dynamic simulation of splashing fluids," in *CA '95: Proceedings of the Computer Animation*, pp. 198–205, 1995.

[93] Osher, S. and Fedkiw, R., *Level set methods and dynamic implicit surfaces*. Springer Verlag, 2002.

[94] Pan, Z., Huang, J., Tong, Y., Zheng, C., and Bao, H., "Interactive localized liquid motion editing," *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, vol. 32, Nov. 2013.

[95] Papazov, C. and Burschka, D., "Deformable 3d shape registration based on local similarity transforms," *Computer Graphics Forum*, vol. 30, no. 5, pp. 1493–1502, 2011.

[96] Park, S. W., Linsen, L., Kreylos, O., Owens, J. D., and Hamann, B., "Discrete sibson interpolation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 243–253, Mar./Apr. 2006.

[97] Parke, F. I. and Waters, K., *Computer Facial Animation*. A. K. Peters, 1996.

[98] Pasko, G., Kravtsov, D., and Pasko, A., "Real-time space-time blending with improved user control," in *Motion in Games*, pp. 146–157, Springer, 2010.

[99] Patel, M. and Taylor, N., "Simple divergence-free fields for artistic simulation," *Journal of Graphics, GPU, and Game Tools*, vol. 10, no. 4, pp. 49–60, 2005.

[100] Pellacini, F., Vidimče, K., Lefohn, A., Mohr, A., Leone, M., and Warren, J., "Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography," *ACM Transactions on Graphics*, vol. 24, pp. 464–470, Aug. 2005.

[101] Perlin, K., "An image synthesizer," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, (New York, NY, USA), pp. 287–296, ACM, 1985.

[102] Pfaff, T., Thuerey, N., Cohen, J., Tariq, S., and Gross, M., "Scalable fluid simulation using anisotropic turbulence particles," in *ACM Transactions on Graphics (TOG)*, vol. 29, p. 174, ACM, 2010.

[103] Pottmann, H., Leopoldseder, S., and Hofer, M., "Registration without icp," *Computer Vision and Image Understanding*, vol. 95, pp. 54–71, 2004.

[104] Rasmussen, N., Enright, D., Nguyen, D., Marino, S., Sumner, N., Geiger, W., Hoon, S., and Fedkiw, R., "Directable photorealistic liquids," in *ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, pp. 193–202, 2004.

[105] Raveendran, K., Thuerey, N., Wojtan, C., and Turk, G., "Controlling liquids using meshes," in *Proce. Symposium on Computer Animation*, SCA '12, pp. 255–264, Eurographics Association, 2012.

[106] Rowley, C., "Model reduction for fluids, using balanced proper orthogonal decomposition," *International Journal of Bifurcation and Chaos*, vol. 15, no. 03, pp. 997–1013, 2005.

[107] Rusinkiewicz, S. and Levoy, M., "Efficient variants of the icp algorithm," in *Proc. 3D Digital Imaging and Modeling*, pp. 145–152, 2001.

[108] Sako, Y. and Fujimura, K., "Shape similarity by homotropic deformation," *The Visual Computer*, vol. 16, no. 1, pp. 47–61, 2000.

[109] Schmid, J., Sumner, R. W., Bowles, H., and Gross, M., "Programmable motion effects," *ACM Trans. Graph.*, vol. 29, pp. 57:1–57:9, July 2010.

[110] Schreiner, J., Asirvatham, A., Praun, E., and Hoppe, H., "Inter-surface mapping," in *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 870–877, ACM, 2004.

[111] Shi, L. and Yu, Y., "Controllable smoke animation with guiding objects," *ACM Transactions on Graphics*, vol. 24, pp. 140–164, January 2005.

[112] Shi, L. and Yu, Y., "Taming liquids for rapidly changing targets," in *ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pp. 229–236, ACM, 2005.

[113] Shinya, M. and Fournier, A., "Stochastic motionmotion under the influence of wind," *Computer Graphics Forum*, vol. 11, no. 3, pp. 119–128, 1992.

[114] Sin, F., Bargteil, A. W., and Hodgins, J. K., "A point-based method for animating incompressible flow," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, (New York, NY, USA), pp. 247–255, ACM, 2009.

[115] Sin, F., Bargteil, A. W., and Hodgins, J. K., "A point-based method for animating incompressible flow," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aug 2009.

[116] Skapin, X. and Lienhardt, P., "Using cartesian product for animation," *The Journal of Visualization and Computer Animation*, vol. 12, no. 3, pp. 131–144, 2001.

[117] Solenthaler, B. and Pajarola, R., "Predictive-corrective incompressible sph," *ACM Trans. Graph.*, vol. 28, pp. 40:1–40:6, July 2009.

[118] Solenthaler, B. and Gross, M., "Two-scale particle simulation," *ACM Trans. Graph.*, vol. 30, pp. 81:1–81:8, July 2011.

[119] Stam, J., "Stable fluids," in *Proc. SIGGRAPH*, pp. 121–128, ACM, 1999.

[120] Stam, J., "Stable fluids," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128, 1999.

[121] Stam, J. and Fiume, E., "Turbulent wind fields for gaseous phenomena," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, (New York, NY, USA), pp. 369–376, ACM, 1993.

[122] Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A., "A material point method for snow simulation," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 32, pp. 102:1–102:10, July 2013.

[123] Sulsky, D., Chen, Z., and Schreyer, H., "A particle method for history-dependent materials," *Computer Methods in Applied Mechanics and Engineering*, vol. 118, pp. 179–196, 1994.

[124] Sumner, R. W., Schmid, J., and Pauly, M., "Embedded deformation for shape manipulation," in *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[125] Szeliski, R., "Matching 3-d anatomical surfaces with non-rigid deformations using octree-splines," *Int. Journal of Computer Vision*, vol. 18, pp. 171–186, 1996.

[126] Taubin, G., "A signal processing approach to fair surface design," in *Proceedings of SIGGRAPH 95*, Annual Conference Series, pp. 351–358, 1995.

[127] Tevs, A., Berner, A., Wand, M., Ihrke, I., Bokeloh, M., Kerber, J., and Seidel, H.-P., "Animation cartographyintrinsic reconstruction of shape and motion," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 2, p. 12, 2012.

[128] Thuerey, N., Keiser, R., Ruede, U., and Pauly, M., "Detail-Preserving Fluid Control," *Symposium on Computer Animation*, pp. 7–12, Jun 2006.

[129] Thürey, N., Keiser, R., Pauly, M., and Rüde, U., "Detail-preserving fluid control," in *ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 7–12, 2006.

[130] Thürey, N., Wojtan, C., Gross, M., and Turk, G., "A multiscale approach to mesh-based surface tension flows," *ACM Transactions on Graphics*, vol. 29, pp. 48:1–48:10, July 2010.

[131] Treuille, A., Lewis, A., and Popović, Z., "Model reduction for real-time fluids," in *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, (New York, NY, USA), pp. 826–834, ACM, 2006.

[132] Turk, G. and O'Brien, J. F., "Shape transformation using variational implicit functions," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, (New York, NY, USA), pp. 335–342, ACM Press/Addison-Wesley Publishing Co., 1999.

[133] Vlasic, D., Baran, I., Matusik, W., and Popović, J., "Articulated mesh animation from multi-view silhouettes," in *ACM Transactions on Graphics (TOG)*, vol. 27, p. 97, ACM, 2008.

[134] von Funck, W., Theisel, H., and Seidel, H.-P., "Vector field based shape deformations," *ACM Trans. Graph*, vol. 25, pp. 1118–1125, 2006.

[135] Warren, J., Schaefer, S., Hirani, A. N., and Desbrun, M., "Barycentric coordinates for convex sets," tech. rep., Advances in Computational and Applied Mathematics, 2004.

[136] Wojtan, C., Thürey, N., Gross, M., and Turk, G., "Deforming Meshes that Split and Merge," *ACM SIGGRAPH 2009 Papers*, vol. 28 (3), p. 9, August 2009.

[137] Wojtan, C., Thürey, N., Gross, M., and Turk, G., "Physics-inspired topology changes for thin fluid features," *ACM Transactions on Graphics*, vol. 29, pp. 50:1–50:8, July 2010.

[138] Wojtan, C. and Turk, G., "Fast viscoelastic behavior with thin features," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–8, 2008.

[139] Yee, Y. L. H., "Spatiotemporal sensistivity and visual attention for efficient rendering of dynamic environments," Master's thesis, Cornell University, 2000.

[140] Yu, J. and Turk, G., "Reconstructing surfaces of particle-based fluids using anisotropic kernels," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 217–225, Eurographics Association, 2010.

[141] Zeng, Y., Wang, C., Wang, Y., Gu, X., Samaras, D., and Paragios, N., "Dense non-rigid surface registration using high-order graph matching," in *IEEE Conf. Computer Vision and Pattern Recognition, CVPR 2010*, pp. 382–389, 2010.

[142] Zhu, Y. and Bridson, R., "Animating sand as a fluid," *ACM Trans. Graph.*, vol. 24, pp. 965–972, July 2005.

VITA

Karthik Raveendran was born in 1985 in Chennai, India. After completing high school in India, he moved to Singapore and graduated with a First Class Honours in computer engineering from Nanyang Technological University in 2007. He then went to Georgia Tech for his graduate studies in computer graphics and animation. He obtained a Masters degree in 2009 while working with Dr. Blair MacIntyre on augmented reality games and a Ph.D. in 2014 for his work on physics based animation under the guidance of Dr. Greg Turk.